NASA CR-172033

Media Independent Interface
Final Report

MS 2-2-5250
SPERRY SPACE SYSTEMS
P.O. BOX 52199
PHOENIX, ARIZONA
85072-2199

Media Independent Interface
Final Report

MS 2-2-5250
SPERRY SPACE SYSTEMS
P.O. BOX 52199
PHOENIX, ARIZONA
85072-2199

CONTENTS

1 INTRODUCTION 1
1.1 PURPOSE 1
1.2 ABBREVIATIONS 1
1.3 DEFINITIONS 1
1.4 DOCUMENTS 2
1.5 MII DEFINITION 2
1.6 MII REQUIREMENTS AND GOALS 4
2 ICD DESCRIPTION 6
2.1 LEVELS OF INTERFACE 6
2.2 FUNCTIONAL INTERFACES 6
2.2.1 MAC-LLC INTERFACE 6
2.2.1.1 LLC-MAC SERVICE PRIMITIVES 7
2.2.2 MAC-SMT INTERFACE 8
2.2.2.1 SYSTEM MANAGEMENT-LLC(SMT-LLC) SERVICE PRIMITIVES AND PARAMETERS 8
2.2.3 INTERFACE CONTROL INFORMATION 10
2.2.3.1 DATA TRANSFER SYNTAX 10
2.2.3.2 ENCODING RULES 11
2.2.4 PROTOCOL DATA UNITS 11
2.2.5 DATA CHANNEL ARCHITECTURE 11
2.2.5.1 DATA CHANNEL DEFINITIONS 11
2.2.5.2 SYNCHRONIZATION 11
2.2.5.3 BUFFER MANAGEMENT 12
2.3 ELECTRICAL/PHYSICAL INTERFACE 12
2.3.1 BUS STANDARDS 13
2.3.2 IMPLEMENTATION OPTIONS 13
2.3.2.1 PHYSICAL PARTITIONING OF FUNCTIONS 13
2.3.2.2 SIMPLE BIU OPTION 14
2.3.2.3 MEDIUM SPEED BIU OPTION 14
2.3.2.4 HIGH SPEED BIU OPTION 17
3 DEMONSTRATION SYSTEM 17
3.1 DEMONSTRATION GOALS 17
3.2 SYSTEM DESCRIPTION 17
3.3 INTERFACE IMPLEMENTATION 19
3.3.1 PRIMITIVES 19
3.3.2 BUFFER MANAGEMENT 19
3.3.3 OPERATING SYSTEM 19
3.3.4 ASN.1 SYNTAX 20
3.4 DEMONSTRATION SYSTEM DETAILED DESCRIPTION . . . 20
3.4.1 COMPONENT IMPLEMENTATION 20
3.4.1.1 TOKEN MAC 20
3.4.1.1.1 MAC PROCESS 21
3.4.1.1.2 INTERRUPT SERVICERS 21
3.4.1.2 STAR*BUS MAC 22
3.4.1.2.1 MAC PROCESS 22

3.4.1.2.2 INTERRUPT SERVICERS 22
3.4.1.3 LLC 23
3.4.1.3.1 SEND 23
3.4.1.3.2 RECEIVE 23
3.4.1.4 STATION MANAGEMENT 23
3.4.1.4.1 COMMAND INTERPRETER 24
3.4.1.4.2 DISPLAY 24
3.4.1.4.3 STATISTICS 24
3.5 DEMONSTRATION TESTING 24

Appendix A OPERATOR'S MANUAL
Appendix B TOKEN BUS/STAR*BUS MEDIUM ACCESS CONTROL SOFTWARE USERS MANUAL

1 INTRODUCTION

1.1 PURPOSE

This report describes the work done on the MII ICD program and makes recommendations based on it. The final output of this study program is an Interface Control Document (ICD) for a Media Independent Interface which is a separate document. This report will provide explanations and rationale for the content of the ICD itself. A basic familiarity with the ISO OSI 7 layer model and the IEEE 802 specifications is very helpful in understanding the contents of this report.

1.2 ABBREVIATIONS

ASN.1 - Abstract Syntax Notation One
CPU - Central Processing Unit
DIS - Draft International Standard
ICD - Interface Control Document
ICI - Interface Control Information
IEEE - Institute of Electrical Electronics Engineers
ISO - Internation Standards Organization
LAN - Local Area Network
LLC - Logical Link Control
MAC - Media Access Control
MII - Media Independent Interface
OSI - Open Systems Interconnection
PDU - Protocol Data Unit
SDU - Service Data Unit
SM - Station Management

1.3 DEFINITIONS

ICI - Interface Control Information - information transferred between adjacent layers to coordinate their joint operation

PDU - Protocol Data Unit - a unit of data specified in a protocol and consisting of protocol information and possibly user data.

SDU - Service Data Unit - an amount of interface data whose identity is preserved from one end of a connection to the other.

1.4 DOCUMENTS

ANSI X3T9/84-100 Fiber Distributed Data Interface (FDDI)
IEEE 802.1 (still draft) Network management
IEEE 802.2 or ISO 8802/2 Logical Link Control (LLC)
IEEE 802.3 or ISO 8802/3 Carrier Sense Media Access (CSMA/CD)
IEEE 802.4 or ISO 8802/4 Token Bus
IEEE 802.5 or ISO 8802/5 Token Ring
IEEE 802.7 or ISO 8802/7 Slotted Ring
ISO DIS 8824 Specification of Abstract Syntax Notation
ISO DIS 8825 Basic Encoding for Abstract Syntax Notation
ISO 7498 ISO OSI Basic Reference Model
ISO 7498 DAD1 connectionless Data Transmission
ISO DP 7498/4 Management Framework
Sperry Report 2055-04 thru 2055-8

1.5 MII DEFINITION

The MII deals with the Data Link Layer, Layer 2 of the ISO 7-Layer communication model. The IEEE 802 specifications divide the Data Link into two sub-layers; the Media Access Control (MAC) is the lower sub-layer and the Logical Link Control (LLC) is the higher sub-layer. The intent of the 802 committee is to provide a single set of LLC functions that supports several different MAC and Physical layer implementations.

The goal of the MII project is to define an interface between these two sub-layers such that all media dependent functions of the Data Link reside in the MAC and all media independent functions reside in the LLC. This division of function results in an interface between the MAC and LLC sub-layers which has been called the Media Independent Interface or MII. The MII concept is illustrated in Figure 1. The object of this division of function is to allow different media access and physical layers to be interfaced to the same upper layer implementation. The intent is to specify this interface to the level such that plug compatible interchangeable modules can be designed independently for both MACs and LLCs. In order to achieve this goal, the further complications added by the Station Management function must be considered. The Station Management entity must communicate with all layers and sub-layers of a given station as it is responsible for

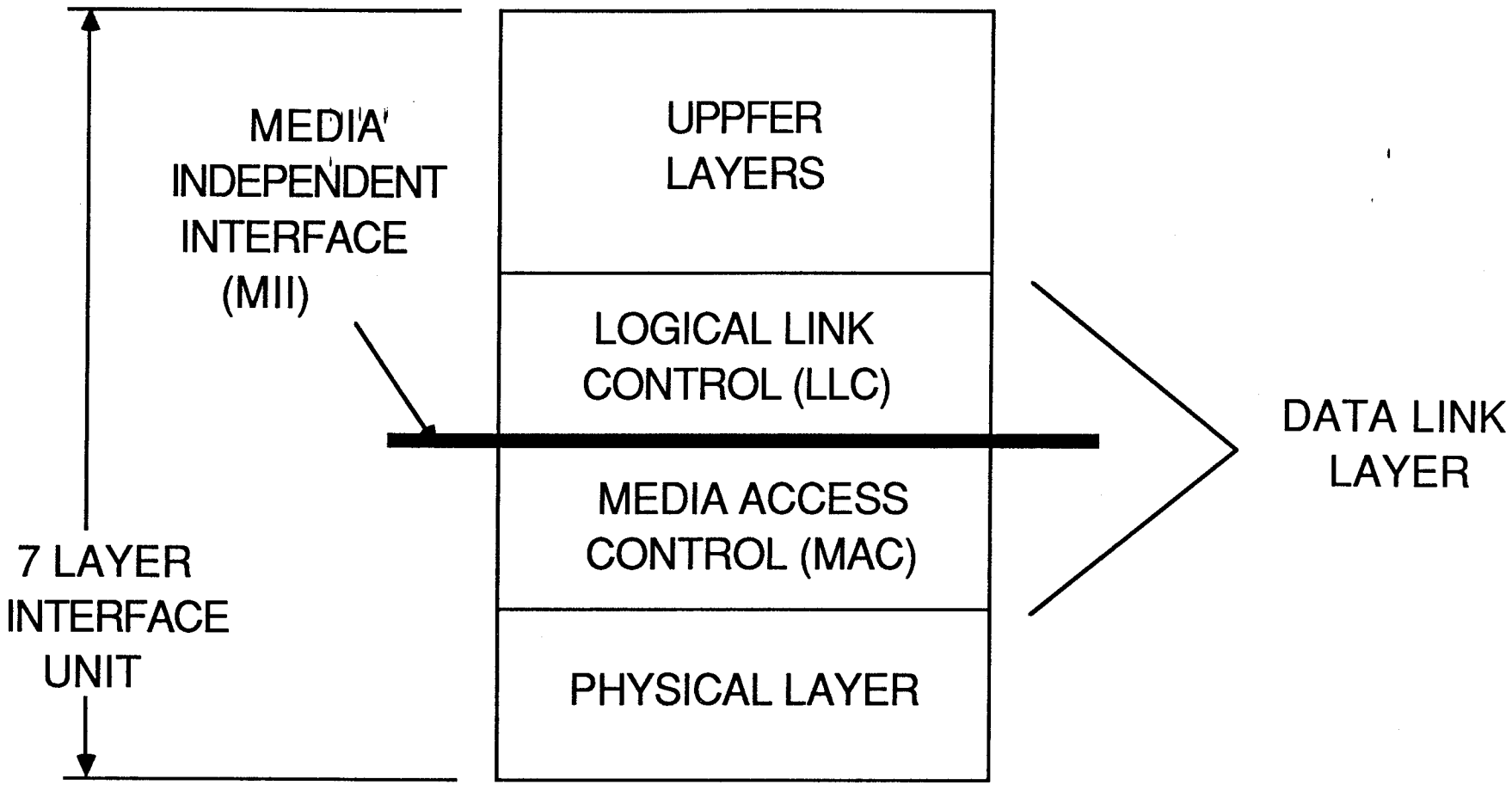

Fig. 1 MEDIA INDEPENDENT INTERFACE

insuring co-operation of all layers within the station and maintaining the health of the station as a whole. It therefore follows that the the interface must not only provide for MAC-LLC communication, but also MAC to Station Management (SMT) communication. Figure 2 shows the location of the entire MII boundary.

1.6 MII REQUIREMENTS AND GOALS

The MII requirements are as follows:

1. Support the functional independence between the interfacing entities, except to the extent of planned interaction through identified (and "controlled") primitives. This feature would make internal changes in the LLC, SM, or MAC transparent to each other.

2. Support asynchronous operation of interfacing entities.

3. Support the operation of high speed LANs, with bit rates on the order of 100 MBPS. Both Star*Bus and FDDI media bit rates are of this magnitude.

4. Support multiples of each type entity, to facilitate bridge, router, and gateway design and multiple port networking.

5. Support ease in reconfiguration, allow for replacing MAC or Physical Layer elements, without modification to the MII or other interfacing elements.

6. Allow for parallel processing and provide for event driven operations in order to support priority transmissions and transfer delay minimization.

7. Use industry accepted standards, especially ISO and IEEE 802.

8. Provide complete interchangeability between MACs.

9. Be expandable yet retain downwards compatibility.

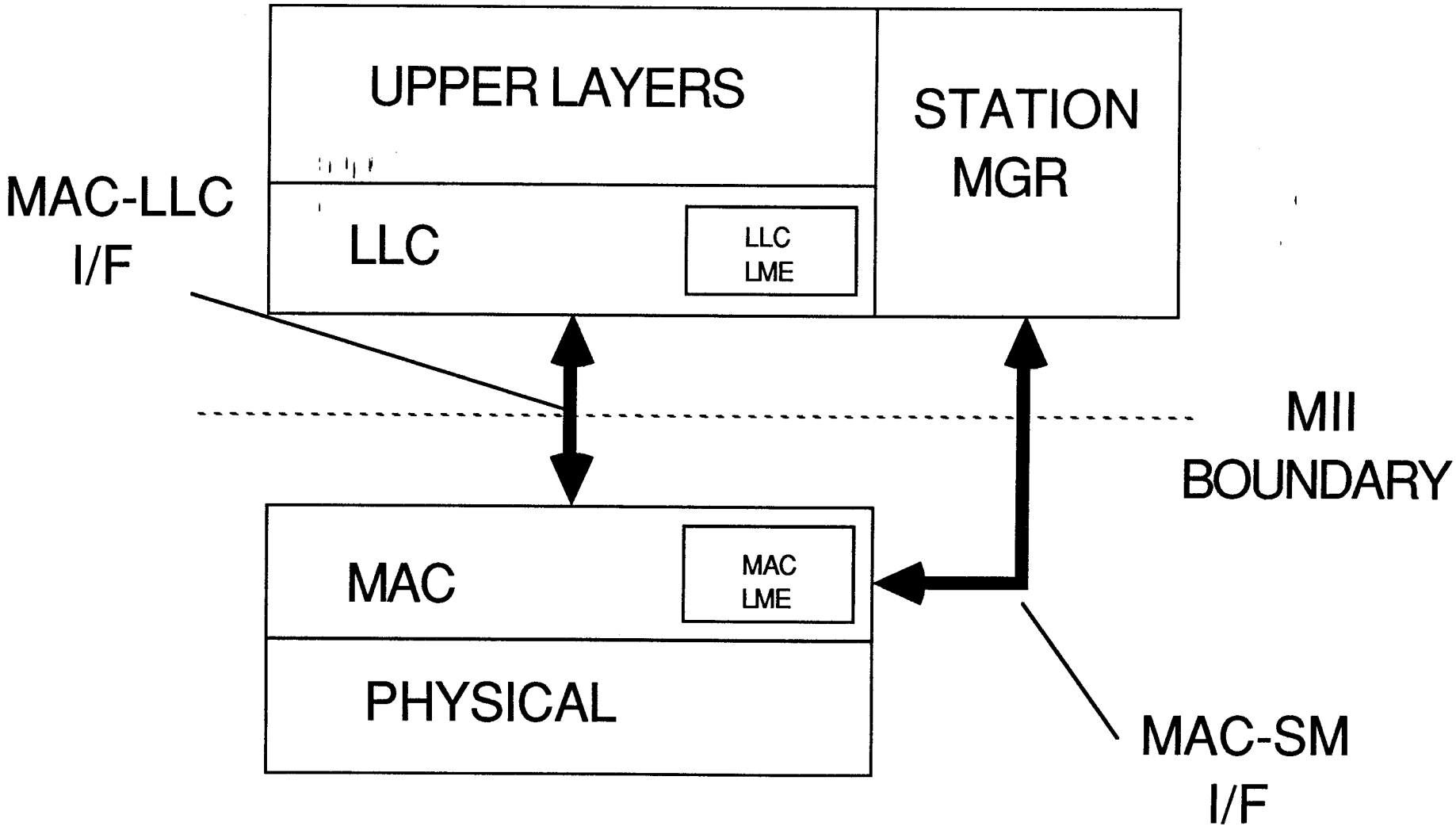

Figure 2 MII BOUNDARY

2 ICD DESCRIPTION

This section describes the contents of the final ICD specification. It is intended to aid in the understanding of the ICD by providing explanations for the features of the architecture and how they relate to the ICD specifications.

2.1 LEVELS OF INTERFACE

The MII deals with three levels of interface: functional, electrical, and physical. The functional level deals with the primitives which are passed between entities and their meanings or interpretations. The design of this level is patterned after the IEEE and ISO specifications. The functions contained in these specifications and the logic contained therein are considered at this level. The electrical interface encompasses all aspects of the bus signals involved, including voltage level, pin assignments, timing specifications,etc. The physical interface deals with form and fit, module dimensions, connector specifications, etc. Each of these levels of the interface will be discussed in detail in the subsequent sections.

2.2 FUNCTIONAL INTERFACES

The functional interface deals with the meaning of the data which crosses the MII boundary. It is involved with what data crosses the boundary and how it is expressed and interpreted. The functional interface includes aspects of both semantics which has to do with meanings or relationship of meanings and syntax which is the form and order or the way in which language elements are connected.

2.2.1 MAC-LLC INTERFACE

The information which must traverse this interface is in the form of data units which are to be sent out or data units which have been received and require processing. This information must be passed in data buffers along with the control information associated with the transfer. Since data is being passed using buffers, a further requirement is to provide information to allow allocation and release of these buffers. Thus, when data has been passed and accepted, a handshake of some type is necessary to let the sender know that the buffer has been processed and can be released or reused. The IEEE specifications provide for the handshake in the case of data transfers from LLC to MAC, but not for data transfers from MAC to LLC, so the ICD has been expanded to provide that function.

The IEEE specifications provide for the handshake in the case of MA_DATA.request but not for MA_DATA.indication, so a primitive has been added to the ICD definition to handle this function.

2.2.1.1 LLC-MAC SERVICE PRIMITIVES

The existing specifications describes primitives and respective parameters that are used to transfer information across the LLC/MAC interface. For a Type 1 LLC, IEEE 802 defines three primitives: MA_DATA.request, MA_DATA.indication, and MA_DATA.confirm. The ICD uses the definitions for these primitives as given in the IEEE specification. A fourth primitive has been added to handle the handshaking requirement for the MA_DATA.indication which has been called MA_DATA.indication_ack. With this augmentation, two pairs of primitives are defined, one pair for each direction of data flow.

The MA_DATA.request and MA_DATA.confirm are use to transfer data from LLC to MAC. The MA_DATA.request primitive is generated by the LLC entity whenever a m sdu (ie MAC service data unit) must be transferred to a peer LLC entity. Upon receiving this primitive the MAC entity appends all MAC specified fields including DA,SA, and any other fields that are unique to the particular media access method in use. The MA_DATA.confirm primitive is generated by the MAC entity in reply to a MA_DATA.request". It is used to indicate the success or failure of the previous associated request. The LLC sub-layer is provided with sufficient information to associate this confirm with the appropriate request.

The MA_DATA.indication and MA_DATA.indication_ack are used to transfer data from MAC to LLC. The MA_DATA.indication primitive is passed from the MAC entity to the LLC entity to indicate the arrival of a frame at the local MAC entity. Frames are only reported if they are validly formatted and their destination address matches that of the local MAC entity. The MA_DATA.indication_ack is a non-IEEE primitive, necessary for proper handling of buffers,etc. The MA_DATA.indication_ack primitive is generated by the LLC entity in acknowledgement of a MA_DATA.indication. It is used to indicate the success or failure of the previous associated .indication. The MAC sub-layer is provided with sufficient information to associate this primitive with the appropriate .indication.

2.2.2 MAC-SMT INTERFACE

This section concentrates on the services provided by the Layer Management Entities (LME's) to the SMAP in order to manage that layer. The SMAP services available to users includes the services provided by the LMEs plus higher level services. Access to the layer management functions is via the Layer Management Interface (LMI), which exists between the local systems management and the layer. The types of services provided across the LMI for the layer are the following:

- The ability for systems management to read and write parameters within the layer
- The ability for systems management to cause actions to occur within the layer.
- The ability for the layer to notify systems management of specific events which have been detected by the layer.

2.2.2.1 SYSTEM MANAGEMENT-LLC(SMT-LLC) SERVICE PRIMITIVES AND PARAMETERS

This section discusses the primitives and their respective parameters used in Systems Management. These primitives are passed across the LMI for the MAC sub-layer in question. This is the more complex of the two interfaces into the MAC. The primitives which traverse this interface are defined in both IEEE 802.1 (System Management Spec) and IEEE 802.4. Unfortunately, the two definitions are inconsistent. The 802.1 definition is used as it is more generally applicable set of primitives and offers more complete functionality. The primitives used for the MII ICD are the following:

- LM_SET_VALUE.invoke
- LM_SET_VALUE.reply
- LM_COMPARE_AND_SET_VALUE.invoke
- LM_COMPARE_AND_SET_VALUE.reply
- LM_GET_VALUE.invoke
- LM_GET_VALUE.reply

- LM_ACTION.invoke
- LM_ACTION.reply
- LM_EVENT.notify
- LM_EVENT.reply

Note that these primitives are arranged as command response pairs with the .reply as a required response for each .invoke or .notify. The reader will notice that the following two primitives constitute somewhat of an exception.

LM_EVENT.notify
LM_EVENT.reply

For this set of primitives, the MAC initiates a .notify and expects a .reply from the SM in response. In all other cases, the SM issues the command primitive and the MAC is require to issue the .reply. This pair of primitives provides the means for the Station manager to set an event mask which allows the MAC to report events without a direct request. The reader will also note that the LM_EVENT.reply is not defined by IEEE. As was the case with the MA_DATA.indication, it was found that a response primitive for the LM_EVENT.nofify was required to allow the proper disposition of message buffers being passed across the MII.

The IEEE 802.4 primitive functions which are implemented for the MII ICD are mapped into the IEEE 802.1 primitives as follows:

IEEE 802.4	IEEE 802.1
MA_INITIALIZE_PROTOCOL.request	LM_ACTION.invoke
MA_INITIALIZE_PROTOCOL.confirm	LM_ACTION.reply
MA_SET_VALUE.request	LM_SET_VALUE.invoke
MA_SET_VALUE.confirm	LM_SET_VALUE.reply
MA_READ_VALUE.request	LM_GET_VALUE.invoke
MA_READ_VALUE.confirm	LM_GET_VALUE.reply
MA_EVENT.indication	LM_EVENT.notify
MA_FAULT_REPORT.indication	LM_EVENT.notify
MA_GROUP_ADDRESS.request	LM_SET_VALUE.invoke
MA_GROUP_ADDRESS.confirm	LM_SET_VALUE.reply
MA_CDATA.request	LM_ACTION.invoke
MA_CDATA.confirm	LM_ACTION.reply
MA_CDATA.indication	LM_EVENT.notify

For detailed explanation of these primitives, the reader is refered to
the ICD.

2.2.3 INTERFACE CONTROL INFORMATION

According to ISO, information transferred between entities in adjacent layers which co-ordinates their joint operation is known as Interface Control Information or ICI. The application data itself is the service data unit or SDU. The combined ICI and SDU is called an interface data unit or IDU. The ICI must be represented in some form which is understood by both layers. To accomplish this in a flexible way, the ICD specifies the use of a data transfer syntax and encoding rules which are also ISO standards.

2.2.3.1 DATA TRANSFER SYNTAX

All data transferred across the MII is described using Abstract Syntax Notation One (ASN.1) per ISO/DIS 8824. This International Standard specifies a notation for defining a syntax, defines a number of simple types, with their tags, and specifies a notation for referencing these types and for specifying values of these types. It further defines mechanisms for contructing new types from more basic types, a specifies a notation for defining such structured types and assigning them tags, and for specifying values of these types. It also defines character sets for use within ASN.1. This notation can be applied whenever it is necessary to define the abstract syntax of information.

Use of this standard syntax in the MII removes any logical inference by order from the interface. Information can be passed in any order and its meaning is inferred by the structure of the message as it is decoded. This results in records which are perhaps more lengthy than absolutely necessary, but offers the advantage of standardization and extensibility in the future. For example, a record containing an address contains information which indicates what kind of address it is as well as the address itself in a variable length format. A receiving entity such as the LLC could be modified to accept an extra long address without violating the standard or compromising compatibility with MAC's using shorter addresses.

2.2.3.2 ENCODING RULES

Although ASN.1 specifies a notation for defining a syntax, it does not specify how data is encoded. Therefore, basic encoding rules for information transfer are supplied by ISO/DIS 8825, Specification of Basic Encoding Rules for ASN.1. This standard defines a set of encoding rules that are applied to values of types defined using ASN.1 which results in transfer syntax for such values. For a detailed definition of the transfer syntax, the reader is referred to the MII ICD itself.

2.2.4 PROTOCOL DATA UNITS

2.2.5 DATA CHANNEL ARCHITECTURE

Communications across the MII boundary is accomplished with predefined data channels. Data is passed over these channels in the form of transactions with linked lists and pointers.

2.2.5.1 DATA CHANNEL DEFINITIONS

2.2.5.2 SYNCHRONIZATION

Each entity has a single input data channel which consists of two globally addressable registers or memory locations. One location is designated as a semaphore. An entity wishing to send a message to the channel must perform a test and set operation on the semaphore location. If the test shows not busy, then the channel was free and the entity has won the right to use the channel. In this case, the entity may then write a pointer to his message in the second register which is reserved for that purpose. The entity receiving the message must have some way of detecting when the pointer has been written. This may be done in a number of ways depending on the system designer. One possible choice is for the receiver to always write a guard word into the pointer prior to resetting the semaphore. Another would be for the act of writing to the pointer register to trigger an interrupt. The method used is immaterial to the sender as long the receiver can properly detect the occurance of the transmission.

At initialization, the station manager must configure the MAC to know the location of the LLC and SMs semaphore bit and link list pointer. Likewise the SM configured the LLC to know the location of the MAC and SMs semaphore bit and link list pointer. Additional status locations are optional (and useful to display self test or BITE during power up).

2.2.5.3 BUFFER MANAGEMENT

In spite of their small size, maintaining memory resources for the ICIs can be difficult since so many are used so often. The IEEE standards do not address this issue, but do provide a reply to MA_Data.request even in a connectionless system. The response or reply for every request is not consistent throughout the IEEE standards and non-existent for the MA_DATA.indication. The lack of a response in this case presents a difficulty in returning memory blocks back to the system for reuse. Therefore the MII ICD requires a reply for every request and each reply will overwrite the memory block which carried the request. In this way, the originator of the request receives his memory block back and may reuse it or return it to a system free pool. This suggests that the total amount of memory a entity requires for ICIs will not exceed the total number of outstanding requests it expects to post. The PDU/SDU data (which is pointed to by a ICI) could work the same way (i.e. a .indication is not acknowledged until the highest layer has copied it) or the receiving entity could copy it and acknowledge it right away. Returning the block to the originator allows for retrys without requiring that an extra copy of the data be held for this purpose. This method is recommended for the MII ICD so as to support both connection and connectionless, to support layer level resets, and ease complications once the data gets to upper layers which block and the records. The entities receive memory from the SM which passes to it a linked list of free fixed size memory segments. The SM supports alarms for low memory and requests for more memory.

In the case of a reset to a layer, data may be lost or corrupted which is allowed per ISO OSI specs. The default for a layer which has been reset is to report the event to the Station Manager. At initialization, the MAC is given a linked list of memory blocks to use for incoming data (from the media) and a linked list of initialized memory blocks for outgoing data (to a remote station). Likewise, the SM will need to perform the same function for the upper layers, but this is transparent to the MII.

2.3 ELECTRICAL/PHYSICAL INTERFACE

2.3.1 BUS STANDARDS

There are several acceptable backplane bus standards available on the market today. Two of the more widely used designs are Multibus II and VMEbus. As would be expect, both of these designs have strong and weak points. Multibus II has built in parity and a large card size. However, it is a synchronous design which tends to be a speed disadvantage in an asynchronous system, and has less bandwidth. It also uses round robin arbitration, which ensures a more even distribution of bandwidth among users, but also limits the burst rate available to a single user.

The VMEbus is an asynchronous design which tends to give it a speed advantage. It is a true multi-master bus, featuring priority driven arbitration. The theoretical bandwidth of 40 Mbyte/sec is larger than that of the Multibus. However, VMEbus uses a smaller card size and does not define any parity lines, although extra undefined lines might be utilized for that purpose.

Either of these buses could be used to support the MII. The selection of the bus standard is a system design issue which impacts performance and design tradeoffs. For example, use of Multibus II bus would require a larger buffer for the media (100 Mbit/sec) because of its relatively low bandwidth. On the other hand, use of Multibus II would prevent the MAC from monopolizing bus bandwidth because of its round robin arbitration scheme.

In order to provide a complete MII a electrical interface must be established. To support further discussion, the VMEbus has been selected as a baseline for the ICD. However, it should be noted that either candidate could be selected. In addition, as little reliance on VME characteristics as possible has been incorporated into the ICD in order to facilitate adoption of another bus standard at a later date.

2.3.2 IMPLEMENTATION OPTIONS

2.3.2.1 PHYSICAL PARTITIONING OF FUNCTIONS

The MII boundary is artificially based on IEEE 802 specifications and the definitions of the primitives. However, the selection of a physical boundary between functions is more of a systems issue than an MII requirement. The decision on where to physically locate functions has great impact on the power, weight, size, and speed factors of the implementation. The ICD does not

dictate the physical partitioning of functions, but leaves these performance related decisions up to the system designer. It is possible for MAC functions to reside totally in a separate physical module or modules than the upper layers, but this is not a requirement. In the case of total physical separation at the MII boundary, the MAC would send ASN messages over the VME bus. Alternatively, messages sent over VME could be proprietary as long as another module could convert them to ASN and retransmit them over the VME to himself. In this way, it is possible to make tradeoffs between hardware complexity and performance all within the confines of the ICD specifications. However, it should be noted that the MII separation does introduce some inefficiencies that are unavoidable.

2.3.2.2 SIMPLE BIU OPTION

The simplest implementation option is to use a single general purpose processor. This option offers the lowest cost at the expense of performance. The majority of the MAC would be implemented using micro-controller technology in order to achieve the speeds necessary for handling the lower level functions. The upper layers, including the LLC would be implemented with software running on the general purpose processor. In order to achieve a minimal hardware implementation, the higher level MAC functions of interfacing to the SM and the LLC could actually be handled with software on the same processor which runs the upper layers. This is the approach that was actually used on the demonstration system. In order to conform with the MII ICD, communications across the MII would be handled over the VME bus, even though the elements communicating are running on the same processor. In order to achieve plug compatibility, the MAC software could be contained in a separate physical ROM which would plug into the processor card. Communications between the lower level MAC functions and the upper layer MAC functions would be handled over the VME bus as well, but would be permitted to use a much simpler language than required for the MII. This implementation option is illustrated in Figure 3.

2.3.2.3 MEDIUM SPEED BIU OPTION

This option offers a compromise between cost and performance. Multiple general purpose processors are used to implement different layers, but all communications takes place over a single VME bus. This option is illustrated in Figure 4. All MAC functions are physically located on a module physically separate from upper layers. Again, lower level MAC functions would be implemented with high-speed

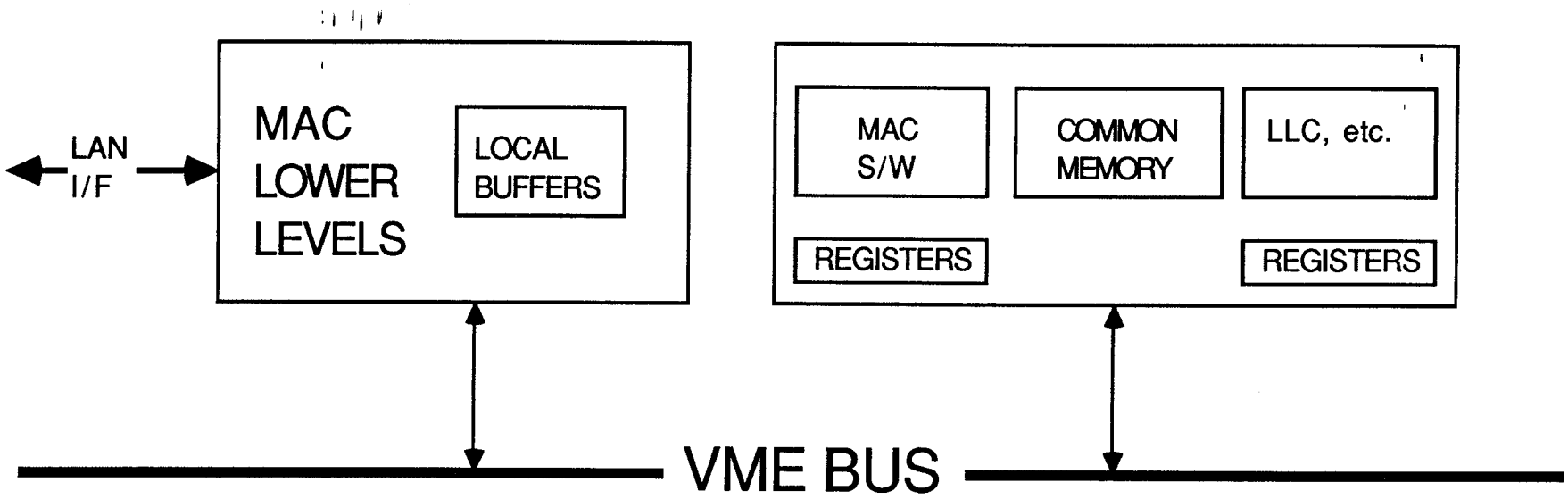

Fig. 3 SIMPLE BIU IMPLEMENATION

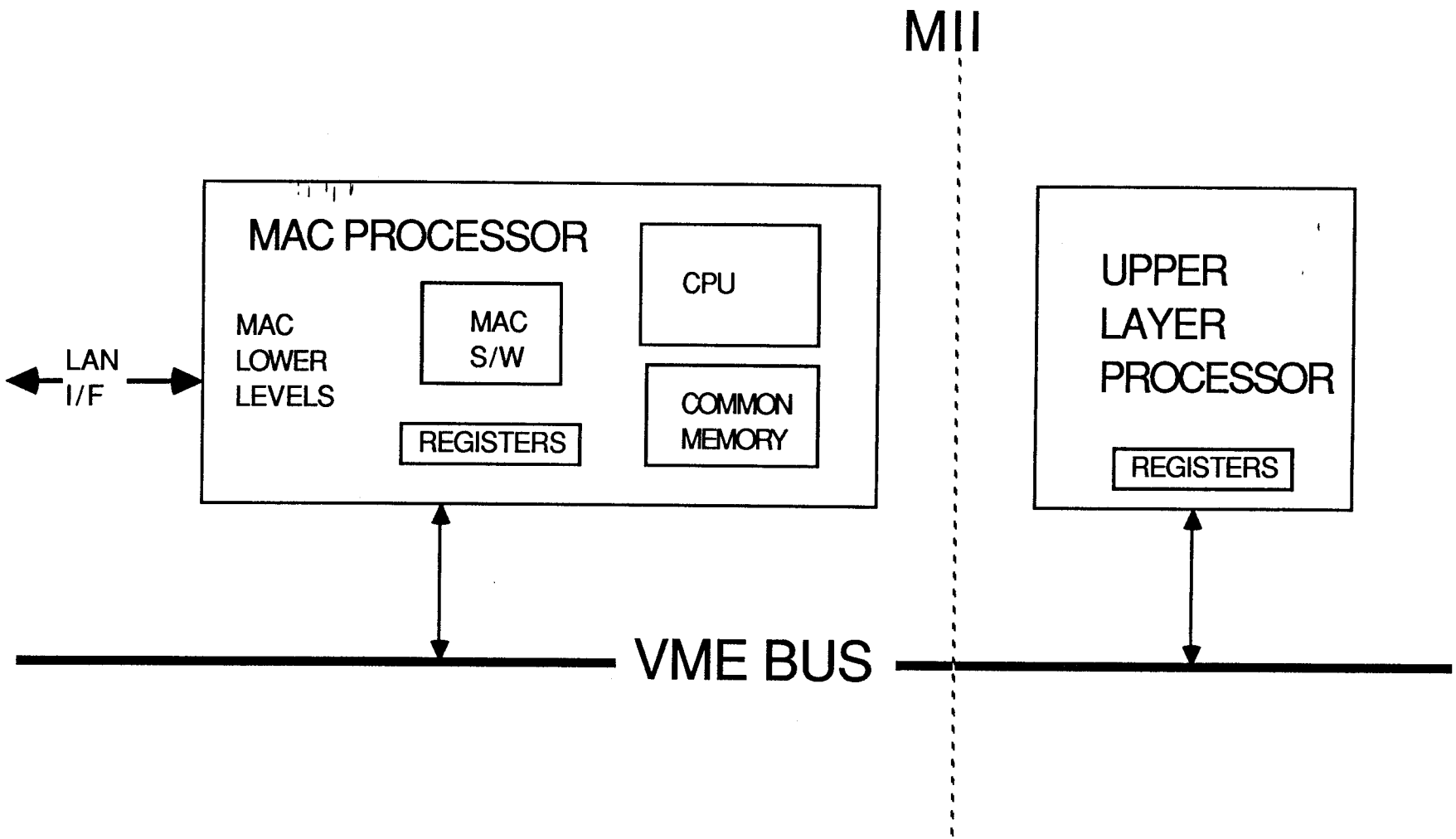

Fig. 4 MEDIUM PERFORMANCE BIU

micro-controller technology. Upper level MAC functions run on their own general purpose processor. However, maximum performance would not be achieved with this design because all data transfers from the LAN interface to the common memory would be performed on the VME bus, tying up considerable bus bandwidth and limiting maximum transfer speeds.

2.3.2.4 HIGH SPEED BIU OPTION

The highest speed operation would be obtained with this option. As with the medium performance option, multiple general purpose processors are used to implement different layers. However, this option would use a VMX bus for high-speed data transfers from the LAN interface to common memory. This design frees the VME bus for other activity, resulting in excellent performance. This option is illustrated in Figure 5.

3 DEMONSTRATION SYSTEM

This section describes the demonstration system which was built in support of the MII development. The demonstration provided a testbed and proving ground for concepts and ideas being incorporated into the ICD.

3.1 DEMONSTRATION GOALS

3.2 SYSTEM DESCRIPTION

The demonstration system was based on existing BIU hardware which was developed under the FODS and NOS contracts. Although this hardware was not designed with the MII requirements in mind, it was adequate to provide the hardware platform for the proof-of-concept work that was required by the ICD project. The BIU consists primarily of an optical interface, a 68000 series processor with DMA, and RS-232 type interfaces. It also includes a high speed parallel interface which was not used for the MII demonstration. New software was written for the BIU processor which tested and demonstrated the MII concepts and designs.

Since a primary goal of the MII is to allow different MAC protocols to interface to a common LLC design, two different MAC implementations were developed for the demonstration. Both MAC's used the same hardware, but different software was used to implement a Star*Bus protocol and a token passing bus protocol. Since both implementations were interfaced to the same upper layer design, a much stronger

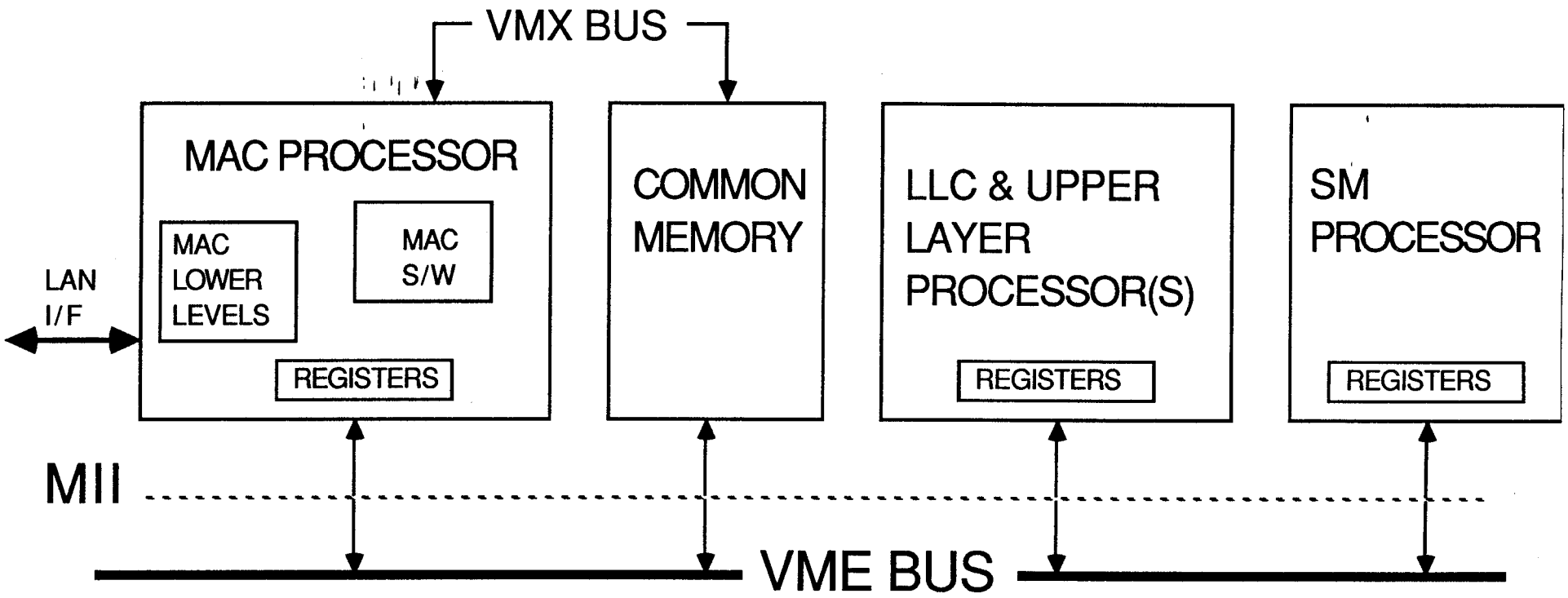

Fig. 5 HIGH PERFORMANCE BIU

demonstration of the MII design resulted.

3.3 INTERFACE IMPLEMENTATION

3.3.1 PRIMITIVES

3.3.2 BUFFER MANAGEMENT

A key element of a communication design is the handling of buffers. The demonstration utilized two basic buffer types for different purposes. Buffers for the data packets themselves (SDU/PDU) were pre-allocated during system initialization. These buffers are of a fixed size large enough to accomodate the maximum size data packet that the bus hardware can handle. These buffers are linked into a free list. Whenever either the MAC or LLC needs a data buffer, it is obtained from this list by a common system service routine. When the buffer is no longer needed, it is returned to the free list, again by a call to a system defined service routine. The buffer includes in its header a pointer and byte count which defines the location of the actual data within the buffer. The header also includes a link which is used to point to the next free buffer in the free list.

Numerous other small buffers are used in the system for passing ICI across the interface. These buffers are obtained via a service call to the memory allocate routine. Generally, only enough memory is allocated to hold the expected message. When these buffers are no longer needed, they are returned to the free memory pool via a system call.

In the demonstration system, these service calls are part of the interface definition. To avoid unnecessary dependencies on operating system implementation, we propose that the task of buffer management and memory allocation be handled by the station manager via standard primitives defined for that purpose. These primitives are defined in the final MII ICD.

3.3.3 OPERATING SYSTEM

The major thrust of the MII ICD project was to demonstrate the separability of the LLC and MAC functions. While the use of separate physical processors for these functions may be necessary to achieve high rates of throughput, it is possible to demonstrate the separability of function required for the MII demonstration with other means. The use of separate software tasks running under a real-time multi-tasking operating system effectively accomplishes the same thing with less hardware. The purpose of a

multi-tasking OS is to allow many independent tasks to run concurrently on one processor. Effectively, the OS transforms one physical processor into multiple logical processors. The technology for doing this is well developed and widely known. This technique was employed for the MII demonstration to show the separation of media dependent and media independent functions.

The operating system services play an important role in the demonstration system. Operating system calls are used to allocate and deallocate memory for the various buffers. In addition, the interprocess communications provisions of the OS are used for communications across the MII. These functions are provided by calls to the signal and wait system service routines. In the final ICD, these communications are replaced by the use of the semaphore and pointer locations defined for each entity. These transfers are expected to occur over the VME bus. The rigorous use of OS services for this function provides the same level of separation as is provided in the final ICD.

3.3.4 ASN.1 SYNTAX

The demonstration system used ASN.1 syntax and encoding rules for data transfers across the MII as described in Section 2. The resultant transfer language is given in the SOFTWARE USER MANUAL in Appendix B.

3.4 DEMONSTRATION SYSTEM DETAILED DESCRIPTION

3.4.1 COMPONENT IMPLEMENTATION

3.4.1.1 TOKEN MAC

The design of the Token Bus MAC was purposely kept very simple for this demonstration. Since it is implemented with software to run on the Star*Bus hardware, performance is inherently limited and no attempt was made to optimize the design. The operation of the Token Bus MAC is very loosely patterned after IEEE 802.3, but many of the advanced features that would normally be designed into firmware were not incorporated. Priority service and ring entry and exit algorithms are not provided. Token recovery is performed via a simple watchdog timer that detects when the token has apparently been lost. Initialization of the token is performed by a manually activated system management command. Unsuccessful attempts to pass the token are detected using the acknowledge supplied by the FODS hardware and retrys are attempted in this case. The Token Bus MAC acknowledges the existence and manipulates the variables in the list below,

although not all of them have any real function in the demonstration system.

VARIABLE NAME	Read	Write	Real	Emulated
TS	x	x		x
NS	x	x	x	
SLOT_TIME	x	x		x
HI_PRI_TOKEN_HOLD_TIME	x	x	x	
MAX_AC_4_ROTATION_TIME	x	x		x
MAX_AC_2_ROTATION_TIME	x	x		x
MAX_AC_0_ROTATION_TIME	x	x		x
MAC_RING_MAINTENANCE_ROTATION_TIME	x	x		x
RING_MAINTENANCE_TIMER_INITIAL_VALUE	x	x		x
MAX_INTER_SOLICIT_COUNT	x	x		x
MIN_POST_SILENCE_PREAMBLE_LENGTH	x	x		x
EVENT_ENABLE_MASK	x	x	x	
MAX_RETRY_LIMIT	x	x	x	
MA_GROUP_ADDRESS	x	x	x	
CHANNEL_ASSIGNMENTS	x	x		x
TRANSMITTED_POWER_LEVEL_ADJUSTMENT	x	x		x
TRANSMITTED_OUTPUT_INHIBITS	x	x		x
RECEIVED_SIGNAL_SOURCES	x	x		x
SIGNALING_MODE	x	x		x
RECEIVED_SIGNAL_LEVEL_REPORTING	x	x		x
LAN_TOPOLOGY_TYPE	x	x		x
MAC_TYPE	x		x	

Variable list, token MAC

3.4.1.1.1 MAC PROCESS

The Token MAC process basically builds and maintains queues of commands and data packets. It processes requests and commands received from the LLC and station manager and builds a queue of data packets for the interrupt servicer to transmit. SM commands are processed by taking appropriate actions and/or building response primitives and sending them back. This process also receives packets from the interrupt service routines and queues them to the LLC.

3.4.1.1.2 INTERRUPT SERVICERS

The interrupt servicers are those routines which deal directly with the hardware. These routines manage the token and perform DMA transfers of data packets from data buffers to the transmitter and from the receiver decoder to data buffers. Data buffers are passed to and from the MAC

process via special queues. Also, these routines collect statistic relating to the passing of tokens.

3.4.1.2 STAR*BUS MAC

The Star*Bus MAC uses a proprietary bus protocol developed for space applications under another project. Most of the protocol is handled in specially designed micro-coded hardware, so that services performed by the software are minimal. The Star*Bus MAC recognizes different variables than the Token MAC, which are listed below.

VARIABLE NAME	Read	Write	Real	Emulated
MAC_TYPE	x		x	
TS	x	x		x
SLOT_TIME	x	x		x
INTER_FRAME_GAP	x	x		x
ATTEMPT_LIMIT	x	x		x
BACK_OFF_LIMIT	x	x		x
JAM_SIZE	x	x		x
MAX_FRAME_SIZE	x	x		x
MIN_FRAME_SIZE	x	x		x
ADDRESS_SIZE	x	x		x
EVENT_ENABLE_MASK	x	x	x	
MA_GROUP_ADDRESS	x	x	x	
MA_GROUP_ADDRESS_ALL				

Variable list, STAR MAC

3.4.1.2.1 MAC PROCESS

The Star*Bus MAC process performs basically the same functions as the Token MAC. However, the Star*Bus MAC must initiate the first transmission after the transmit queue has been allowed to empty since there is no token being passed to cause data transmissions to begin. Also, the Star*Bus MAC recognizes and processes variables which have meaning to its own protocol

3.4.1.2.2 INTERRUPT SERVICERS

The Star*Bus MAC interrupt servicers perform basically the same functions as those of the Token MAC. However, there is no token to manage which is the major difference.

3.4.1.3 LLC

As discussed in a previous section, development of a fully functional LLC was found unnecessary to adequately demonstrate the MII design. Therefore, the demonstration LLC component includes only the basic functions of sending and receiving packets. The special LLC test functions were not required for the demonstration. Basically, the LLC must deal with only the four primitives defined for the MAC-LLC interface. This section of software also included test functions of packet generation and analysis.

3.4.1.3.1 SEND

This process is responsible for handling all data packets to be transmitted by the MAC, whether they are generated internally or entered from the console. Packets may be generated in a number of ways and from various sources which are described in more detail in the Operators Manual in the appendix. For each packet sent, this process builds a request primitive and passes it to the MAC. It also handles the reply primitive which is passed back as a result of each request and disposes of the associated data buffer as is appropriate. This process also keeps statistics on the number of packets which have been originated and successfully sent to the MAC for transmission.

3.4.1.3.2 RECEIVE

The receive process collects and disposes of packets received from other stations. It accepts MA_DATA.indication primitives from the MAC and generates MA_DATA.indication_ack for each one received, thus returning the data buffer to the sender(MAC). It also collects statistics on received packets.

3.4.1.4 STATION MANAGEMENT

A formal station manager was outside of the requirements for the ICD program and therefore was not include in the demonstration. In essence, the test operator performs the function of station manager in the demonstration system. This portion of the system provides the operator with a control and test interface into each BIU via an RS-232 interface. The software translates operator input commands into the appropriate SM primitives to perform the desired actions. Statistics information gathered throughout the system are also collected and displayed per operator commands.

3.4.1.4.1 COMMAND INTERPRETER

This process performs all work associated with handling input from the operator. This includes buffering of command lines, editing functions, parsing of the command into keywords and associated parameters, and calling of appropriate handlers to perform or initiate requested functions. The command interpreter accepts user input commands, translates them into SM primitives which are sent to the MAC. It also accepts test commands and activates the requested test functions, for example the sending of packets or activation of a display mode.

3.4.1.4.2 DISPLAY

This process provides display of packet contents on the CRT. It is required because the operating system provides only blocking I/O which stops the process requesting the I/O until the request has been satisfied. Therefore, this process is used to perform the display function so that other processes may continue to execute while the output proceeds.

3.4.1.4.3 STATISTICS

This process drives the real time statistics display for the system when activated. It also serves as a collection point for SM events reported by the MAC process. The statistics display is updated periodically per a system alarm timer.

3.5 DEMONSTRATION TESTING

The Test Set Up which was used to perform the ICD MII demonstration consists of 3 BIU's communicating over a fiber optic link (Star*Bus). Each of the BIU's interfaces to a CRT terminal via an RS-232. The terminal provides the operator with test, control, and station management functions for the interfacing BIU. A block diagram of the test setup is shown in Figure 6.

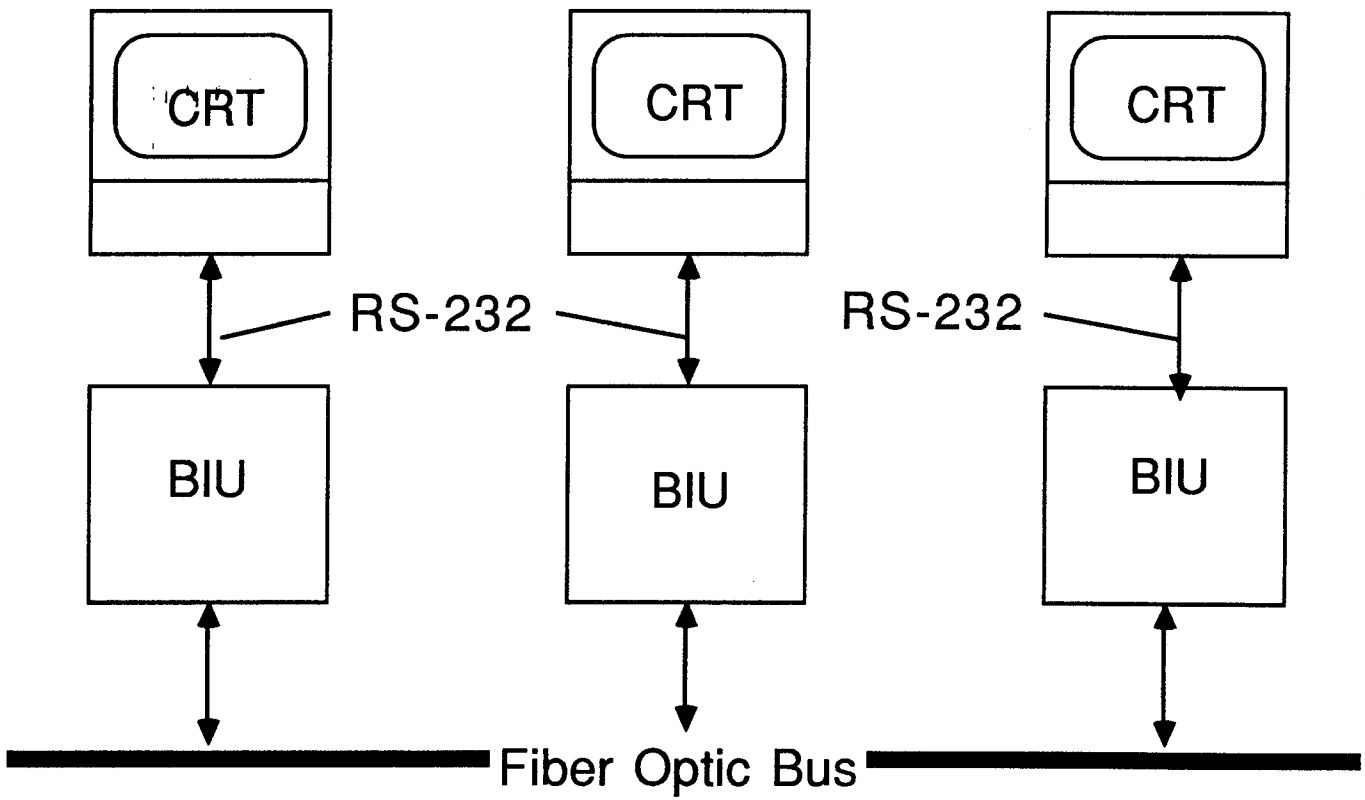

FIG. 6 DEMONSTRATION SETUP

Appendix A
Operator's Manual

MS 2-2-5250
SPERRY SPACE SYSTEMS
P.O. BOX 52199
PHOENIX, ARIZONA
85072-2199

User Console Commands

The following are valid commands which the operator may enter from the console and which result in the described activity or functions. Lists of valid variables which may be manipulated for the two MAC's are found in the Token Bus/Star*Bus Medium Access Control Software User Manual.

SET varname - examine variable and allow change

DIS_VAR varname - display value of single variable

TEST varname1,test,varname2,const - perform comparison test specified between two variables and set the first variable equal to a constant if the test is satisfied. The parameter "test" may take on the following values:

‹ - test for varname1 less than varname2

› - test for varname1 greater than varname2

= - test for varname1 equal to varname2

‹› - test for varname1 not equal to varname2

‹= - test for varname1 less than or equal to varname2

›= - test for varname1 greater than or equal to varname2

The comparison is an atomic test performed using the compare and set Station Management primitive. No action other than display of an error message will be taken if either variable is not a valid MAC variable. However, type-checking will only be performed by the MAC when the primitive is processed.

CTEST varname1,test,constant - perform comparison test specified between a variable and a constant and set the variable equal to the constant if the test is satisfied. The parameter "test" may take on the following values:

‹ - test for varname1 less than constant

› - test for varname1 greater than constant

= - test for varname1 equal to constant

Appendix B

Appendix B

<> - test for varnamel not equal to constant

<= - test for varnamel less than or equal to constant

>= - test for varnamel greater than or equal to constant

The comparison is an atomic test performed using the compare and set Station Management primitive. No action other than display of an error message will be taken if the variable is not a valid MAC variable. However, type-checking of the variable and constant will only be performed by the MAC when the primitive is processed.

DIS_MAC - display all MAC variables

SENDS [address][,pattern] - send single packets to station address starting with pattern as random seed for packet generator. Once entered, the routine accepts:

s - send next packet, rotating pattern

r - resend last packet

x - quit

c - send console defined packet

Both parameters are optional and will default to previously used address and pattern.

SENDC [t][,][p][,][address list] - send packets continuously

t = time between packets

p = (r)otating or (f)ixed pattern or (c)onsole entered packet or fixed (q)ueue of rotating packets.

address list = list of up to 16 destination addresses separated by commas.

KILL_SEND - cancel the continuous sending of packets previously activated via the SENDC command.

KILL_STATS - cancel the display of real-time statistics.

CONSOLE [addr] - accept and send ASCII text from the console and display received packets on console in ASCII

DEF_PACKET - enter a packet from the keyboard in HEX for subsequent transmission by another command.

DIS_STATS - display static statistics

RESET - re-initialize variables in this station

DIS_RT - display realtime statistics in the realtime window.

FREEZE - freeze MAC activity

UNFREEZE - resume normal MAC activity

ENTER_RING - activate the ring by passing a token frame to the next station. This command will also activate the watchdog token timer so that this station will generate a new token if it is dropped.

EXIT_RING - consume the token by refusing to pass the next token received and deactivate the watchdog token timer.

DIS_GA - display group address table

GRP_ADDR [-]addr - modify group address table by adding or deleting the specified address. The minus sign causes the address to be deleted from the table if it already exists. If table is full or address is already defined or attempt is made to delete address not previously defined, then an appropriate error message is displayed.

CLR_GA - clear the entire group address table.

CLR_STATS - clear all statistical counters.

DIS_PACKET x - display packet in HEX format on the CRT. Parameter x may take on values of:

r = last received packet

s = last sent packet

TOKEN BUS / STAR BUS
MEDIUM ACCESS CONTROL

SOFTWARE USERS MANUAL

MS 2-2-5250
SPERRY SPACE SYSTEMS
P.O. BOX 52199
PHOENIX, ARIZONA
85072-2199

CONTENTS

1	PURPOSE	2
2	SCOPE	2
2.1	GENERAL	2
3	ARCHITECTURE	3
4	INTERFACES	4
4.1	LLC	4
4.1.1	COMMANDS	4
4.1.2	SYNTAX	5
4.2	STATION MANAGER COMMANDS	12
4.2.1	COMMANDS	12
4.2.2	IEEE 802.3 COMMAND DESCRIPTIONS	13
4.2.3	SYNTAX	25
4.2.4	FORMAL SYNTAX SPECIFICATION	25
4.2.5	IEEE 802.4 SM COMMAND DESCRIPTIONS	33
4.2.6	SYNTAX	48
4.2.7	FORMAL SYNTAX SPECIFICATION	48

1 PURPOSE

The purpose of this document is to give interface descriptions for implementation of the Media Access Controller (MAC) and software which is to interface to it.

2 SCOPE

This document contains general descriptions of interface syntax, commands and variables and responses the mac will have to those commands and variables.

2.1 GENERAL

Each section of this document contains a subsection which describes with text the general theory of operation for its respective component.

For additional information see the ISO DIS 8824 document Specification of Abstract Syntax Notation One (ASN.1).

3 ARCHITECTURE

The Media Access Controller (MAC) is the device which manages the actual transfer of data on a local area network (LAN). It interfaces to the physical layer (i.e. the hardware) and manages it in such a way as to provide a means by which it can share the communication resources the hardware provides.

The MAC task has two interfaces to the outside world; the Logical Link Control (LLC), and the Station Manager (SM).

The LLC is the source and sink of data to the MAC. It requests the MAC to perform a service of shipping data to an address. In addition the MAC will indicate to the LLC when another remote LLC has passed data to it.

The instructions to the MAC are queued by the LLC and a message is sent to the MAC so it will be awakened by the OS. The MAC de-queues it and puts it into a message queue for the MACs interrupt routine. The interrupt routines take it from there. The MAC can get a message from the interrupt routines which indicate to the MAC that data has arrived from another remote MAC. The MAC queues a message to the LLC to awaken it and indicate that a message has arrived and where it is.

The Station Manager interfaces in the same way as the LLC but its primitives are move extensive. It has the ability to set variables in the MAC, read its status, reset it, and perform all the necessary functions to manage the media.

For each request passed to the MAC from the LLC or SM there is a confirm. When the MAC initiates a request, event, or indication, it expects a confirm. These confirms allow the requestors to deallocate the original message.

Each section gives further details on the method by which the MAC commands operate.

4 INTERFACES

4.1 LLC

4.1.1 COMMANDS -

The LLC Interface supports MA_DATA request and confirms and MA_DATA indication and indi_acks. Details of these commands are described in the IEEE 802.3, IEEE 802.4 and IEEE 802.2 documents. A brief explanation follows;

```
MA_DATA.REQUEST
{ DESTINATION_ADDRESS, M_SDU, DESIRED_QUALITY }
```

This command comes from the LLC to the MAC and represents a request to ship the data pointed to by the M_SDU to the station at address DESTINATION_ADDRESS using a level of quality of DESIRED_QUALITY. The MAC is expected to respond with the following command;

```
MA_DATA.CONFIRMATION
{ QUALITY, STATUS }
```

This command from the MAC to the LLC will indicate to the LLC that the data previously requested to be shipped has been sent. The LLC knows what data was sent because the MA_DATA.CONFIRMATION message is returned to the LLC by overwriting the original MA_DATA.REQUEST message.

```
MA_DATA.INDICATION
{ DESTINATION_ADDRESS, SOURCE_ADDRESS,
  M_SDU, QUALITY                        }
```

This command from the MAC to the LLC will indicate to the LLC that data located at pointer M_SDU from the station SOURCE_ADDRESS was sent to DESTINATION_ADDRESS (needed to identify when a group address is used) with a QUALITY of service. The MAC expects the LLC to overwrite the MA_DATA.INDICATION with the MA_DATA.INDI_ACK thus allowing it to release the message buffer.

```
MA_DATA.INDI_ACK
{ STATUS }
```

This command from the LLC to the MAC will indicate to the MAC that the LLC has no more use for the indicate message buffer.

4.1.2 SYNTAX -

The LLC communicates to the MAC across the MII. The syntax of such communication is described in the Software Design Document according to Abstract Syntax Notation One or ASN.1 (ISO DIS 8824). The information described is encoded to the basic coding rules as found in ASN.1 (ISO DIS 8825). Some sample records follow the syntax notations in the station management interface section.

```
MESSAGE_RECORD ::= [PRIVATE 0] CHOICE
{ MA_Data_request  [0]  ma_request_type    |
  MA_Data_confirm  [1]  ma_confirm_type    |
  MA_Data_indicate [2]  ma_indicate-type   |
  MA_Indi_ack      [3]  ma_indi_ack_type    }

ma_request_type ::= SET
{ destination_address [0] net_address_type           ,
  M_SDU               [1] M_SDU_type                 ,
  requested_Ser_class [2] req_ser_type               ,
  frame_control       [3] frame_con_type   (optional) ,
  stream              [4] stream_type      (optional) ,
  link_list           [5] link_list_type   (optional) ,
  token_class         [6] token_class_type (optional)
}
  -- Multiple request_info's are allowed for FDDI only.
  -- token_class allowed for FDDI only and has a default.



ma_indicate-type ::= SET
{ destination_address [0] net_address_type           ,
  source_address      [1] net_address_type           ,
  M_SDU               [2] M_SDU_type                 ,
  reception_status    [3] rec_status (optional)      ,
  requested_Ser_class [4] req_ser_type (optional)    ,
  frame_control       [5] frame_con_type   (optional) ,
}
  -- The optional parameters have a default value.


ma_confirm_type ::=
{  transmit_status       [0]  tran_status                ,
   provided_ser_class    [1]  provided_ser_type (optional),
   number_of_sdu_links   [2]  number_of_sdu (optional)
}

ma_indi_ack_type ::= SET
{ indi_status    [0] integer -- 1 = good 0= not accepted
 }

net_address_type ::= CHOICE
{  net_add  VALUE_INTEGER_1 }

M_SDU_type ::= SET
{  SDU_PTR     [0]   ADDRESS,
   SDU_SIZE    [1]   INTEGER,
   buff_num    [2]   INTEGER  }
```

```
req_ser_type ::= SET
{ priority       [0]   INTEGER, (optional)
  response       [1]   INTEGER, (optional) -- ack = 1
  quality_of_ser [2]   INTEGER } (optional)
```

```
provided_ser_type ::= SET
{ priority       [0]   INTEGER, (optional)
  response       [1]   INTEGER, (optional) -- ack = 1
  quality_of_ser [2]   INTEGER  (optional) }

indi_ack_type ::= SET
{ indi_ack_status  ack_status }

frame_con_type ::= SET {} --TBD

link_list_type ::= SET {} --TBD

stream_type ::= SET {} --TBD

token_class_type ::= SET {} --TBD

rec_status ::= CHOICE
{  status [0] INTEGER -- 1 = good }

tran_status ::= CHOICE
{  status [0] INTEGER -- 1 = good }

number_of_sdu ::= CHOICE {} --TBD
```

WHAT FOLLOWS IS WHAT IS ACTUALLY USED OUT OF THE ABOVE
FOR A REQUEST (TO A 802.3 MAC):

```
MESSAGE_RECORD ::= [PRIVATE 0] CHOICE
{MA_Data_request  [0]
   {destination_address [0]
       { net_add  VALUE_INTEGER_1
       }
   M_SDU                [1]
       {SDU_PTR      [0]   ADDRESS,              -- IROB
       }
   requested_Ser_class [2]
    {priority         [0]   INTEGER, (optional) -- priority
     response         [1]   INTEGER, (optional) -- ack = 1
    }
   }
 }
```

Typical confirm of a previous response.

```
MESSAGE_RECORD ::= [PRIVATE 0]
{ MA_Data_confirm  [1]
    {transmit_status        [0]
                  {status [0] INTEGER -- 1 = good
                  }
     provided_ser_class     [1]
                  {priority        [0]   INTEGER, --priority
                  }
    }
}
```

A typical indicate message from the MAC to the LLC.

```
MESSAGE_RECORD ::= [PRIVATE 0]
{ MA_Data_indicate [2]
   { destination_address [0]
              { net_add  VALUE_INTEGER_1 }              ,
     source_address       [1]
              { net_add  VALUE_INTEGER_1 }              ,
     M_SDU                [2]
              { SDU_PTR      [0]   ADDRESS  -- IROB prt  ,
              }
     reception_status     [3]
              { status [0] INTEGER -- 1 = good
              }                                         ,
     requested_Ser_class [4]
              { priority        [0]   INTEGER, -- priority
              }
   }
}
```

```
MESSAGE_RECORD ::= [PRIVATE 0] CHOICE
{   MA_Indi_ack    [3]
     { indi_status    [0] integer -- 1 = ok 0= !ok,
     }
}
```

ENCODING:

```
e0 1c              MESSAGE_RECORD
  e0 1a               MA_Data_request
    e0 06
      c0 02 xx xx        net_add INTEGER
    e1 06
      c0 04 xx xx xx xx SDU_PTR ADDRESS
    e2 08
      c0 02 xx xx        priority INTEGER
      c1 02 xx xx        response INTEGER


e0 0e              MESSAGE_RECORD
  e1 0c               MA_Data_confirm
    e0 04
      c0 02 xx xx        status INTEGER
    e0 04
      c0 02 xx xx        priority INTEGER


e0 22              MESSAGE_RECORD
  e2 20               MA_Data_indicate
    e0 04                destination
      c0 02 xx xx          net_add  INTEGER
    e1 04                source
      c0 02 xx xx          net_add  INTEGER
    e2 06
      c0 04 xx xx xx xx   SDU_PTR ADDRESS
   e3 04
      c0 02 xx xx          status INTEGER
   e4 04
      c0 02 xx xx          priority INTEGER


e0 08              MESSAGE_RECORD
  e3 06               MA_Indi_ack
    c0 02 xx xx           indi_status INTEGER
```

4.2 STATION MANAGER COMMANDS

4.2.1 COMMANDS -

The Station manager sends invoke commands to the MAC and the MAC responds with a reply response. The pairs which follow are first station manager command followed by the MAC response.

SM_MAC_LM_SET_VALUE.INVOKE
SM_MAC_LM_SET_VALUE.REPLY

SM_MAC_LM_GET_VALUE.INVOKE
SM_MAC_LM_GET_VALUE.REPLY

SM_MAC_LM_COMPARE_AND_SET_VALUE.INVOKE
SM_MAC_LM_COMPARE_AND_SET_VALUE.REPLY

SM_MAC_ACTION_VALUE.INVOKE
SM_MAC_ACTION_VALUE.REPLY

The Station manager can set an event mask which allows the MAC to report events without a direct request. The MAC initiates a NOTIFY and expects a REPLY from the LLC in response.

SM_MAC_EVENT_VALUE.NOTIFY
SM_MAC_EVENT_VALUE.REPLY

4.2.2 IEEE 802.3 COMMAND DESCRIPTIONS -

SM_MAC_LM_SET_VALUE.INVOKE
{ PARAMETER_TYPE, ACCESS_CONTROL_INFO }

The objective of the SM_MAC_LM_SET_VALUE.INVOKE command by the SM is to set a value in the MAC as defined by the parameter_type structure. This structure specifies both the variable to be set and the value to which it is set.

SM_MAC_LM_SET_VALUE.REPLY
{ STATUS }

The objective of the reply by the MAC to the SM is to indicate the success or failure of a previous SM_MAC_LM_SET_VALUE.INVOKE. The SM expects the MAC to overwrite the SM_MAC_LM_SET_VALUE.INVOKE with the SM_MAC_LM_SET_VALUE.REPLY thus allowing the SM to release the message buffer.

SM_MAC_LM_GET_VALUE.INVOKE
{ PARAMETER_TYPE, ACCESS_CONTROL_INFO }

The objective of the SM_MAC_LM_GET_VALUE.INVOKE command by the SM is to get a value in the MAC as defined by the parameter_type structure. This structure specifies the variable to be read.

SM_MAC_LM_GET_VALUE.REPLY
{ PARAMETER_TYPE, STATUS }

The objective of the reply by the MAC to the SM is to indicate the success or failure of a previous SM_MAC_LM_GET_VALUE.INVOKE. The SM expects the MAC to overwrite the SM_MAC_LM_GET_VALUE.INVOKE with the SM_MAC_LM_GET_VALUE.REPLY thus allowing the SM to release the message buffer.

SM_MAC_LM_COMPARE_AND_SET_VALUE.INVOKE
{ PARAMETER_TYPE, OPERATION_COMMAND,
ACCESS_CONTROL_INFO }

The Compare and Set value command forces the MAC to do a comparison (of either a given constant or of a MAC variable) against a MAC variable. If the comparison is true then the MAC variable is over written. The PARAMETER_TYPE indicates the parameter to be over written and the value to use. The OPERATION_COMMAND structure specifies the comparison to do, and the constant or MAC variable to use in the comparison.

SM_MAC_LM_COMPARE_AND_SET_VALUE.REPLY
{ STATUS, RETURN_VAL }

The objective of the reply by the MAC to the SM is to indicate the success or failure of a previous SM_MAC_LM_COMPARE_AND_SET_VALUE.INVOKE . The SM expects the MAC to overwrite the SM_MAC_LM_COMPARE_AND_SET_VALUE.INVOKE with the SM_MAC_LM_COMPARE_AND_SET_VALUE.REPLY thus allowing the SM to release the message buffer.

SM_MAC_ACTION_VALUE.INVOKE
{ PARAMETER_ID, ACCESS_CONTROL_INFO }

The objective of the SM_MAC_ACTION_VALUE.INVOKE command by the SM is to force a MAC operation (i.e. reset, freeze) in the MAC as defined by the parameter_ID structure. This structure specifies the action to be performed.

SM_MAC_ACTION_VALUE.REPLY
{ STATUS, ACTION_REPORT }

The objective of the reply by the MAC to the SM is to indicate the success or failure of a previous SM_MAC_ACTION_VALUE.INVOKE . The SM expects the MAC to overwrite the SM_MAC_ACTION_VALUE.INVOKE with the SM_MAC_ACTION_VALUE.REPLY thus allowing the SM to release the message buffer.

MAC_SM_EVENT_VALUE.NOTIFY
{ EVENT_ID }

The objective of the MAC_SM_EVENT_VALUE.NOTIFY command by the MAC is to report a event which has occurred in the MAC as defined by the EVENT_ID structure. This structure specifies the Event and an integer. These events can be masked by setting the EVENT_MASK variable.

MAC_SM_EVENT_VALUE.REPLY
{ STATUS }

The objective of the following reply by the SM to the MAC is to indicate the success or failure of a previous MAC_SM_EVENT_VALUE.NOTIFY. The MAC expects the SM to overwrite the MAC_SM_EVENT_VALUE.NOTIFY with the MAC_SM_EVENT_VALUE.REPLY thus allowing the MAC to release the message buffer.

```
READ_WRITE_VALUE_TYPES ::=  CHOICE       {
   [0]       MAC_TYPE                     |
   [1]       TS                           |
   [2]       SLOT_TIME                    |
   [3]       INTER_FRAME_GAP              |
   [4]       ATTEMPT_LIMIT                |
   [5]       BACK_OFF_LIMIT               |
   [6]       JAM_SIZE                     |
   [7]       MAX_FRAME_SIZE               |
   [8]       MIN_FRAME_SIZE               |
   [9]       ADDRESS_SIZE                 |
   [10]      EVENT_ENABLE_MASK            |
   [11]      MA_GROUP_ADDRESS             |
   [12]      MA_GROUP_ADDRESS_ALL           }
```

TS ::= VALUE_ADDRESS_1

This variable represents the address of this station. Since FODS has its address set in hardware this variable has no effect on MAC performance.

SLOT_TIME ::= VALUE_INTEGER_1

This variable represents the slot time of this station. This is the maximum time this station must wait on another station to respond to a transmition. The FODS knows this as T_Gap. Since FODS has T_Gap set in hardware this variable has no effect on MAC performance.

EVENT_ENABLE_MASK ::= EVENT_ENABLE_BITS

```
EVENT_ENABLE_BITS ::= BIT STRING
{ DUPLICATE_ADDRESS                 (2),
  FAULTY_TRANSMITTER                (3),
  XMIT_QUEUE_THRESHOLD_EXCEEDED     (4),
  RECEIVE_QUEUE_THRESHOLD_EXCEEDED  (5),
  WATCH_DOG_TIMEOUT                 (6),
  FROZEN                            (7),
  MAX_RETRY_ENCOUNTERED             (10),
  BAD_MESSAGE_SENT                  (11)  }
                        -- Where 1 is enabled
```

The MAC will report events when discovered and the appropriate bit is set in the MASK above. These bits are inspected each time the event has occurred and the MAC task is active. The event is reported only once whenever the actual occurence is detected.

ATTEMPT_LIMIT ::= VALUE_INTEGER_1

This is the maximum number of times that a packet will be retransmitted when the acknowledgement indicates a bad transmittion. Since this is a connectionless system this variable will not be used.

MA_GROUP_ADDRESS ::= VALUE_ADDRESS_1

The MAC can respond to a group of group addresses. This is one of two methods for the Station Manager to tell the MAC which addresses are acceptable. This implementation will

support a total of 16 group addresses. A positive value in this command will set this address as part of the group addresses (unless there all used up) and a negative address will delete this address from the table.

MA_GROUP_ADDRESS_ALL ::= VALUE_ADDRESS_16

The MAC can respond to a group of group addresses. This is one of two methods for the Station Manager to tell the MAC which addresses are acceptable. This implementation will support a total of 16 group addresses. This command will write or read all 16 addresses at once. Addresses which are not valid addresses should be set to a negative one. Therefore only positive addresses will be passed in with this command unless there a negative one (a null address).

FREEZE_MAC ::= VALUE_INTEGER_1

This variable when set to one will freeze the MAC from taking any data from the local LLC. A negative one will unfreeze it and allow any queued messages to be processed. This will cause a burst effect and may cause loss of data due to the connectionless nature of the system and the limited buffer space.

MAC_TYPE ::= 03h

This variable is a read only variable and indicates which version of MAC is responding.

INTER_FRAME_GAP ::= VALUE_INTEGER_1

This is function cannot be changed with software in FODs and this variable will not have a effect on the performance of the MAC.

BACK_OFF_LIMIT ::= VALUE_INTEGER_1

This is function cannot be changed with software in FODs and this variable will not have a effect on the performance of the MAC.

JAM_SIZE ::= VALUE_INTEGER_1

This is function cannot be changed with software in FODs and this variable will not have a effect on the performance

of the MAC.

MAX_FRAME_SIZE ::= VALUE_INTEGER_1

This is function cannot be changed with software in FODs and this variable will not have a effect on the performance of the MAC.

MIN_FRAME_SIZE ::= VALUE_INTEGER_1

This is function cannot be changed with software in FODs and this variable will not have a effect on the performance of the MAC.

ADDRESS_SIZE ::= VALUE_INTEGER_1

This is function cannot be changed with software in FODs and this variable will not have a effect on the performance of the MAC.

```
STATUS_TYPE ::= _CHOICE
{     UNDEFINED_ERROR            [0]   VALUE_INTEGER_1 |
      SUCCESS                    [1]   VALUE_INTEGER_1 |
      REFUSE_TO_COMPLY           [2]   VALUE_INTEGER_1 |
      NOT_SUPPORTED              [3]   VALUE_INTEGER_1 |
      ERROR_IN_PERFOR            [4]   VALUE_INTEGER_1 |
      NOT_AVAILABLE              [5]   VALUE_INTEGER_1 |
      BAD_PARAMETER_ID           [6]   VALUE_INTEGER_1 |
      BAD_PARAMETER_OPERATION    [7]   VALUE_INTEGER_1 |
      BAD_PARAMETER_VALUE        [8]   VALUE_INTEGER_1 |
      BAD_EXPECTED_VALUE         [9]   VALUE_INTEGER_1  }
```

These are responses to a command indicating the status of the command. Following are expected uses of these responses;

```
UNDEFINED_ERROR - Request was not understood or no
                  appropriate error message available.
SUCCESS - A successful operation has been completed.
REFUSE_TO_COMPLY - The operation was impossible or illegal.
NOT_SUPPORTED - The operation is not supported or
                recognized.
ERROR_IN_PERFOR - A error was encountered
                  during operation.
NOT_AVAILABLE - Information is not yet
                available.
BAD_PARAMETER_ID - Parameter ID was not
                   recognized.
BAD_PARAMETER_OPERATION - Operation requested
                          was not recognized
BAD_PARAMETER_VALUE - The Parameter
                      value was bad.
BAD_EXPECTED_VALUE  - The expected value was
                      illegal.
```

```
EVENT_TYPES ::= IMPLICIT SEQUENCE
{ EVENT_CLASS-        EVENT_CLASS_TYPES     }

     EVENT_CLASS_TYPES ::=  CHOICE
     { LOCAL      [0]    EVENT_IDENTIFIER_TYPES      |
       REMOTE     [1]    EVENT_IDENTIFIER_TYPES       }
```

Events in this implementation are always LOCAL (as opposed to events that occurred in a remote node).

```
EVENT_IDENTIFIER_TYPES ::= CHOICE
{ DUPLICATE_ADDRESS                [2]  VALUE_INTEGER_1  |
```

```
  FAULTY_TRANSMITTER                [3]  VALUE_INTEGER_1  |
  XMIT_QUEUE_THRESHOLD_EXCEEDED     [4]  VALUE_INTEGER_1  |
  RECEIVE_QUEUE_THRESHOLD_EXCEEDED  [5]  VALUE_INTEGER_1  |
  WATCH_DOG_TIMEOUT                 [6]  VALUE_INTEGER_1  |
  FROZEN                            [7]  VALUE_INTEGER_1  |
  MAX_RETRY_ENCOUNTERED             [10] VALUE_INTEGER_1  |
  BAD_MESSAGE_SENT                  [11] VALUE_ADDRESS_1  }
```

These events are reported upon the discovery of the following conditions;

DUPLICATE_ADDRESS - Does nothing since FODs doesn't report other addresses.

FAULTY_TRANSMITTER - Does nothing since FODs doesn't report bad transmitters.

XMIT_QUEUE_THRESHOLD_EXCEEDED - Flagged when the MAC cannot get buffer space for outgoing data.

RECEIVE_QUEUE_THRESHOLD_EXCEEDED - Flagged when the MAC cannot get buffer space for incoming data.

WATCH_DOG_TIMEOUT - Flagged if the hardware watch dog timer expires.

FROZEN - Flagged when the MAC is frozen. Reported only once.

MAX_RETRY_ENCOUNTERED - Flagged when a the max retry is encountered. Any IRL4 interrupt indicates the hardware retryed beyond the retry limit.

BAD_MESSAGE_SENT -Flagged when the MAC discovers a message which does not agrees with its indicated structure size (i.e. bad length field).

```
ACTION_VALUE_TYPES ::= CHOICE
{ RESET              [0]   VALUE_INTEGER_1          |
  FREEZE/UNFREEZE    [1]   FREEZE_MAC               }
```

The ACTION_VALUE_TYPES allow the following;

Reset value_integer_1 = anything:

A reset will flush all queues, set all operating parameters to their initial values, lose the token (if its holding it), and await work from either the media or the LLC.

FREEZE/UNFREEZE FREEZE_MAC = 1 will freeze mac.
= -1 will unfreeze mac.

A FREEZE/UNFREEZE command with Freeze option will make the MAC main task ignore all queues but the SM. In effect the MAC is frozen to local service only. Packets arriving from remote nodes and from the LLC will be queued until the buffer is exceeded or the MAC is unfrozen. The token is still passed as normal. Commands from the Station Manager are processed while frozen.

```
OPERATION_COMMAND_TYPES ::= CHOICE
  {TEST_<<              [0]  READ_WRITE_VALUE_TYPES    |
   TEST_>>              [1]  READ_WRITE_VALUE_TYPES    |
   TEST_==              [2]  READ_WRITE_VALUE_TYPES    |
   TEST_<>              [3]  READ_WRITE_VALUE_TYPES    |
   TEST_<=              [4]  READ_WRITE_VALUE_TYPES    |
   TEST_>=              [5]  READ_WRITE_VALUE_TYPES    |
   <<_GIVEN_CONSTANT    [6]  GIVEN                     |
   >>_GIVEN_CONSTANT    [7]  GIVEN                     |
   ==_GIVEN_CONSTANT    [8]  GIVEN                     |
   <>_GIVEN_CONSTANT    [9]  GIVEN                     |
   <=_GIVEN_CONSTANT    [10] GIVEN                     |
   >=_GIVEN_CONSTANT    [11] GIVEN                       }
```

The above operations expects a variable (we'll call var1) to be internal. The complete structure includes either a variable or constant which we'll call var2. The constant is used to overwrite Var1 in case the operation test true so in the case of two internal vars being tested a constant is also passed in. The above operation commands imply the following:

```
TEST_<<  - if var1 << var2 then var1=constant
TEST_>>  - if var1 >> var2 then var1=constant
TEST_==  - if var1 == var2 then var1=constant
TEST_<>  - if var1 <> var2 then var1=constant
TEST_<=  - if var1 <= var2 then var1=constant
TEST_>=  - if var1 >= var2 then var1=constant
<<_GIVEN_CONSTANT - if var1 << constant then var1=constant
>>_GIVEN_CONSTANT - if var1 >> constant then var1=constant
==_GIVEN_CONSTANT - if var1 == constant then var1=constant
```

<>_GIVEN_CONSTANT - if var1 <> constant then var1=constant
<=_GIVEN_CONSTANT - if var1 <= constant then var1=constant
>=_GIVEN_CONSTANT - if var1 <= constant then var1=constant

Var1 is a MAC parameter to be tested (internal). Its value is always returned along with a status. Var2 is a MAC parameter (internal) or a constant (external) used in the comparison of Var1 (internal). Var1 always refers to a variable located in the MAC. Var2 is either located in the MAC (a compare of two internal variables) or as a constant (external) passed in. In all cases a true test forces Var1 to be a external constant.

CONSTANT ::= VALUE_INTEGER_1

VALUE_INTEGER_1 ::= IMPLICIT LONG_WORD

VALUE_ADDRESS_1 ::= IMPLICIT LONG_WORD (32 BITS)

VALUE_ADDRESS_16 ::= IMPLICIT ARRAY OF 16 LONG_WORDS
(32 BITS EACH)

4.2.3 SYNTAX - STATION MANAGER INTERFACE SYNTAX

The station manager communicates to the MAC across the MII. The syntax of such communication is described below according to Abstract Syntax Notation One or ASN.1 (ISO DIS 8824). The information described is encoded to the basic coding rules as found in ASN.1 (ISO DIS 8825). Some sample records follow the syntax notations.

4.2.4 FORMAL SYNTAX SPECIFICATION -

```
MESSAGE_RECORD ::= [PRIVATE 0] CHOICE
{ [0]  SM_MAC_LM_SET_VALUE.INVOKE                          |
  [1]  SM_MAC_LM_SET_VALUE.REPLY                           |
  [2]  SM_MAC_LM_GET_VALUE.INVOKE                          |
  [3]  SM_MAC_LM_GET_VALUE.REPLY                           |
  [4]  SM_MAC_LM_COMPARE_AND_SET_VALUE.INVOKE              |
  [5]  SM_MAC_LM_COMPARE_AND_SET_VALUE.REPLY               |
  [6]  SM_MAC_ACTION_VALUE.INVOKE                          |
  [7]  SM_MAC_ACTION_VALUE.REPLY                           |
  [8]  SM_MAC_EVENT_VALUE.NOTIFY                           |
  [9]  SM_MAC_EVENT_VALUE.REPLY                          }


SM_MAC_LM_SET_VALUE.INVOKE ::= IMPLICIT SEQUENCE
{ PARAMETER_TYPE         READ_WRITE_VALUE_TYPES ,
  ACCESS_CONTROL_INFO    NULL                   }

SM_MAC_LM_SET_VALUE.REPLY ::= IMPLICIT SEQUENCE
{ RETURN_VAL         READ_WRITE_VALUE_TYPES,
  STATUS                 STATUS_TYPE}

SM_MAC_LM_GET_VALUE.INVOKE ::= IMPLICIT SEQUENCE
{ PARAMETER_TYPE         READ_WRITE_VALUE_TYPES ,
  ACCESS_CONTROL_INFO    NULL                   }

SM_MAC_LM_GET_VALUE.REPLY ::= IMPLICIT SEQUENCE
{ PARAMETER_TYPE     READ_WRITE_VALUE_TYPES ,
  STATUS             STATUS_TYPE }


SM_MAC_LM_COMPARE_AND_SET_VALUE.INVOKE ::= IMPLICIT SEQUENCE
{ PARAMETER_TYPE         DUMMY_RW_TYPES,
  OPERATION_COMMAND      OPERATION_COMMAND_TYPES,
  ACCESS_CONTROL_INFO    NULL                   }

SM_MAC_LM_COMPARE_AND_SET_VALUE.REPLY ::= IMPLICIT SEQUENCE
{  RETURN_VAL            READ_WRITE_VALUE_TYPES,
   STATUS                STATUS_TYPE }


SM_MAC_ACTION_VALUE.INVOKE ::= IMPLICIT SEQUENCE
{ PARAMETER_ID              ACTION_VALUE_TYPES ,
  ACCESS_CONTROL_INFO       NULL}

SM_MAC_ACTION_VALUE.REPLY ::= IMPLICIT SEQUENCE
{ STATUS                STATUS_TYPE,
  ACTION_REPORT         NULL     }
```

```
MAC_SM_EVENT_VALUE.NOTIFY ::= IMPLICIT SEQUENCE
{ EVENT_ID                  EVENT_TYPES }

MAC_SM_EVENT_VALUE.REPLY ::= IMPLICIT SEQUENCE
{ STATUS                    STATUS_TYPE }
```

```
READ_WRITE_VALUE_TYPES ::=  CHOICE       {
   [0]       MAC_TYPE                     |
   [1]       TS                           |
   [2]       SLOT_TIME                    |
   [3]       INTER_FRAME_GAP              |
   [4]       ATTEMPT_LIMIT                |
   [5]       BACK_OFF_LIMIT               |
   [6]       JAM_SIZE                     |
   [7]       MAX_FRAME_SIZE               |
   [8]       MIN_FRAME_SIZE               |
   [9]       ADDRESS_SIZE                 |
   [10]      EVENT_ENABLE_MASK            |
   [11]      MA_GROUP_ADDRESS             |
   [12]      MA_GROUP_ADDRESS_ALL           }
```

```
DUMMY_RW_TYPES ::=  CHOICE        {
MAC_TYPE                              [0] VALUE_INTEGER_1   |
TS                                    [1] VALUE_ADDRESS_1   |
SLOT_TIME                             [2] VALUE_INTEGER_1   |
INTER_FRAME_GAP                       [3] VALUE_INTEGER_1   |
ATTEMPT_LIMIT                         [4] VALUE_INTEGER_1   |
BACK_OFF_LIMIT                        [5] VALUE_INTEGER_1   |
JAM_SIZE                              [6] VALUE_INTEGER_1   |
MAX_FRAME_SIZE                        [7] VALUE_INTEGER_1   |
MIN_FRAME_SIZE                        [8] VALUE_INTEGER_1   |
ADDRESS_SIZE                          [9] VALUE_INTEGER_1   |
EVENT_ENABLE_MASK                     [10] VALUE_INTEGER_1  |
MA_GROUP_ADDRESS                      [12] VALUE_INTEGER_1  |
MA_GROUP_ADDRESS_ALL                  [13] VALUE_INTEGER_1  |
```

TS ::= VALUE_ADDRESS_1

NS ::= VALUE_ADDRESS_1

SLOT_TIME ::= VALUE_INTEGER_1

EVENT_ENABLE_MASK ::= EVENT_ENABLE_BITS

MA_GROUP_ADDRESS ::= VALUE_ADDRESS_1

MA_GROUP_ADDRESS_ALL ::= VALUE_ADDRESS_16

FREEZE_MAC ::= VALUE_INTEGER_1

MAC_TYPE ::= 03h

```
STATUS_TYPE ::= CHOICE
{    UNDEFINED_ERROR            [0]   VALUE_INTEGER_1 |
     SUCCESS                    [1]   VALUE_INTEGER_1 |
     REFUSE_TO_COMPLY           [2]   VALUE_INTEGER_1 |
     NOT_SUPPORTED              [3]   VALUE_INTEGER_1 |
     ERROR_IN_PREFOR            [4]   VALUE_INTEGER_1 |
     NOT_AVAILABLE              [5]   VALUE_INTEGER_1 |
     BAD_PARAMETER_ID           [6]   VALUE_INTEGER_1 |
     BAD_PARAMETER_OPERERATION  [7]   VALUE_INTEGER_1 |
     BAD_PARAMETER_VALUE        [8]   VALUE_INTEGER_1 |
     BAD_EXPECTED_VALUE         [9]   VALUE_INTEGER_1  }

EVENT_TYPES ::= IMPLICIT SEQUENCE
{ EVENT_CLASS          EVENT_CLASS_TYPES     }

EVENT_CLASS_TYPES ::=  CHOICE
{ LOCAL      [0]    EVENT_IDENTIFIER_TYPES          |
  REMOTE     [1]    EVENT_IDENTIFIER_TYPES              }

EVENT_IDENTIFIER_TYPES ::= CHOICE
{ DUPLICATE_ADDRESS                 [2]  VALUE_INTEGER_1   |
  FAULTY_TRANSMITTER                [3]  VALUE_INTEGER_1   |
  XMIT_QUEUE_THRESHOLD_EXCEEDED     [4]  VALUE_INTEGER_1   |
  RECEIVE_QUEUE_THRESHOLD_EXCEEDED  [5]  VALUE_INTEGER_1   |
  WATCH_DOG_TIMEOUT                 [6]  VALUE_INTEGER_1   |
  FROZEN                            [7]  VALUE_INTEGER_1   |
  MAX_RETRY_ENCOUNTERED             [10] VALUE_INTEGER_1   |
  BAD_MESSAGE_SENT                  [11] VALUE_ADDRESS_1    }


EVENT_ENABLE_BITS ::= BIT STRING
{ DUPLICATE_ADDRESS                 (2),
  FAULTY_TRANSMITTER                (3),
  XMIT_QUEUE_THRESHOLD_EXCEEDED     (4),
  RECEIVE_QUEUE_THRESHOLD_EXCEEDED  (5),
  WATCH_DOG_TIMEOUT                 (6),
  FROZEN                            (7),
  MAX_RETRY_ENCOUNTERED             (10)
  BAD_MESSAGE_SENT                  (11) }
                        -- Where 1 is enabled


ACTION_VALUE_TYPES ::= CHOICE
{ RESET              [0]   VALUE_INTEGER_1          |
  FREEZE/UNFREEZE    [1]   FREEZE_MAC               }


OPERATION_COMMAND_TYPES ::= CHOICE
  {TEST_<<           [0]  DUMMY_RW_TYPES            |
   TEST_>>           [1]  DUMMY_RW_TYPES            |
```

```
   TEST_==              [2]  DUMMY_RW_TYPES          |
   TEST_<>              [3]  DUMMY_RW_TYPES          |
   TEST_<=              [4]  DUMMY_RW_TYPES          |
   TEST_>=              [5]  DUMMY_RW_TYPES          |
   <<_GIVEN_CONSTANT    [6]  GIVEN                   |
   >>_GIVEN_CONSTANT    [7]  GIVEN                   |
   ==_GIVEN_CONSTANT    [8]  GIVEN                   |
   <>_GIVEN_CONSTANT    [9]  GIVEN                   |
   <=_GIVEN_CONSTANT    [10] GIVEN                   |
   >=_GIVEN_CONSTANT    [11] GIVEN                    }


GIVEN ::= CHOICE
  { [0] VALUE_INTEGER_1       |
    [1] VALUE_ADDRESS_1        }

CONSTANT ::=  VALUE_INTEGER_1

VALUE_INTEGER_1 ::= IMPLICIT INTEGER

VALUE_ADDRESS_1 ::=  IMPLICIT LONG_WORD (32 BITS)

VALUE_ADDRESS_16 ::=  IMPLICIT ARRAY OF 16 LONG_WORDS
                                (32 BITS EACH)
```

4.2.5 IEEE 802.4 SM COMMAND DESCRIPTIONS -

SM_MAC_LM_SET_VALUE.INVOKE
{ PARAMETER_TYPE, ACCESS_CONTROL_INFO }

The objective of the SM_MAC_LM_SET_VALUE.INVOKE command by the SM is to set a value in the MAC as defined by the parameter_type structure. This structure specifies both the variable to be set and the value to which it is set.

SM_MAC_LM_SET_VALUE.REPLY
{ STATUS }

The objective of the reply by the MAC to the SM is to indicate the success or failure of a previous SM_MAC_LM_SET_VALUE.INVOKE. The SM expects the MAC to overwrite the SM_MAC_LM_SET_VALUE.INVOKE with the SM_MAC_LM_SET_VALUE.REPLY thus allowing the SM to release the message buffer.

SM_MAC_LM_GET_VALUE.INVOKE
{ PARAMETER_TYPE, ACCESS_CONTROL_INFO }

The objective of the SM_MAC_LM_GET_VALUE.INVOKE command by the SM is to get a value in the MAC as defined by the parameter_type structure. This structure specifies the variable to be read.

SM_MAC_LM_GET_VALUE.REPLY
{ PARAMETER_TYPE, STATUS }

The objective of the reply by the MAC to the SM is to indicate the success or failure of a previous SM_MAC_LM_GET_VALUE.INVOKE. The SM expects the MAC to overwrite the SM_MAC_LM_GET_VALUE.INVOKE with the SM_MAC_LM_GET_VALUE.REPLY thus allowing the SM to release the message buffer.

SM_MAC_LM_COMPARE_AND_SET_VALUE.INVOKE
{ PARAMETER_TYPE, OPERATION_COMMAND,
ACCESS_CONTROL_INFO }

The Compare and Set value command forces the MAC to do a

comparison (of either a given constant or of a MAC variable) against a MAC variable. If the comparison is true then the MAC variable is over written. The PARAMETER_TYPE indicates the parameter to be over written and the value to use. The OPERATION_COMMAND structure specifies the comparison to do, and the constant or MAC variable to use in the comparison.

SM_MAC_LM_COMPARE_AND_SET_VALUE.REPLY
{ STATUS, RETURN_VAL }

The objective of the reply by the MAC to the SM is to indicate the success or failure of a previous SM_MAC_LM_COMPARE_AND_SET_VALUE.INVOKE . The SM expects the MAC to overwrite the SM_MAC_LM_COMPARE_AND_SET_VALUE.INVOKE with the SM_MAC_LM_COMPARE_AND_SET_VALUE.REPLY thus allowing the SM to release the message buffer.

SM_MAC_ACTION_VALUE.INVOKE
{ PARAMETER_ID, ACCESS_CONTROL_INFO }

The objective of the SM_MAC_ACTION_VALUE.INVOKE command by the SM is to force a MAC operation (i.e. reset, freeze) in the MAC as defined by the parameter_ID structure. This structure specifies the action to be performed.

SM_MAC_ACTION_VALUE.REPLY
{ STATUS, ACTION_REPORT }

The objective of the reply by the MAC to the SM is to indicate the success or failure of a previous SM_MAC_ACTION_VALUE.INVOKE . The SM expects the MAC to overwrite the SM_MAC_ACTION_VALUE.INVOKE with the SM_MAC_ACTION_VALUE.REPLY thus allowing the SM to release the message buffer.

MAC_SM_EVENT_VALUE.NOTIFY
{ EVENT_ID }

The objective of the MAC_SM_EVENT_VALUE.NOTIFY command by the MAC is to report a event which has occurred in the MAC as defined by the EVENT_ID structure. This structure specifies the Event and an integer. These events can be masked by setting the EVENT_MASK variable.

MAC_SM_EVENT_VALUE.REPLY
{ STATUS }

The objective of the following reply by the SM to the MAC is to indicate the success or failure of a previous MAC_SM_EVENT_VALUE.NOTIFY. The MAC expects the SM to overwrite the MAC_SM_EVENT_VALUE.NOTIFY with the MAC_SM_EVENT_VALUE.REPLY thus allowing the MAC to release the message buffer.

Variables which can be accessed are;

VARIABLE NAME	Read	Write	Real	Emulated
TS	x	x		x
NS	x	x	x	
SLOT_TIME	x	x		x
HI_PRI_TOKEN_HOLD_TIME	x	x	x	
MAX_AC_4_ROTATION_TIME	x	x		x
MAX_AC_2_ROTATION_TIME	x	x		x
MAX_AC_0_ROTATION_TIME	x	x		x
MAC_RING_MAINTENANCE_ROTATION_TIME	x	x		x
RING_MAINTENANCE_TIMER_INITIAL_VALUE	x	x		x
MAX_INTER_SOLICIT_COUNT	x	x		x
MIN_POST_SILENCE_PREAMBLE_LENGTH	x	x		x
EVENT_ENABLE_MASK	x	x	x	
MAX_RETRY_LIMIT	x	x	x	
MA_GROUP_ADDRESS	x	x	x	
CHANNEL_ASSIGNMENTS	x	x		x
TRANSMITTED_POWER_LEVEL_ADJUSTMENT	x	x		x
TRANSMITTED_OUTPUT_INHIBITS	x	x		x
RECEIVED_SIGNAL_SOURCES	x	x		x
SIGNALING_MODE	x	x		x
RECEIVED_SIGNAL_LEVEL_REPORTING	x	x		x
LAN_TOPOLOGY_TYPE	x	x		x
MAC_TYPE	x		x	

```
READ_WRITE_VALUE_TYPES ::=  CHOICE        {
   [0]        MAC_TYPE                                    |
   [1]        NS                                          |
   [2]        SLOT_TIME                                   |
   [3]        HI_PRI_TOKEN_HOLD_TIME                      |
   [4]        MAX_AC_4_ROTATION_TIME                      |
   [5]        MAX_AC_2_ROTATION_TIME                      |
   [6]        MAX_AC_0_ROTATION_TIME                      |
   [7]        MAC_RING_MAINTENANCE_ROTATION_TIME          |
   [8]        RING_MAINTENANCE_TIMER_INITIAL_VALUE        |
   [9]        MAX_INTER_SOLICIT_COUNT                     |
   [10]       MIN_POST_SILENCE_PREAMBLE_LENGTH            |
   [11]       EVENT_ENABLE_MASK                           |
   [12]       MAX_RETRY_LIMIT                             |
   [13]       MA_GROUP_ADDRESS                            |
   [14]       MA_GROUP_ADDRESS_ALL                        |
   [15]       CHANNEL_ASSIGNMENTS                         |
   [16]       TRANSMITTED_POWER_LEVEL_ADJUSTMENT          |
   [17]       TRANSMITTED_OUTPUT_INHIBITS                 |
   [18]       RECEIVED_SIGNAL_SOURCES                     |
   [19]       SIGNALING_MODE                              |
   [20]       RECEIVED_SIGNAL_LEVEL_REPORTING             |
   [21]       LAN_TOPOLOGY_TYPE                           |
   [22]       TS                                            }
```

TS ::= VALUE_ADDRESS_1

This variable represents the address of this station. Since FODS has its address set in hardware this variable has no effect on MAC performance.

NS ::= VALUE_ADDRESS_1

This variable represents the address of the next station. The IEEE 802.4 would normally calculate this address however in this implementation this address must be set in order for the MAC to know where the next station is in order to pass the token.

SLOT_TIME ::= VALUE_INTEGER_1

This variable represents the slot time of this station. This is the maximum time this station must wait on another station to respond to a transmition. The FODS knows this as T_Gap. Since FODS has T_Gap set in hardware this variable has no effect on MAC performance.

HI_PRI_TOKEN_HOLD_TIME ::= VALUE_INTEGER_1

This variable represents the token hold time of this station. This is the maximum time this station can hang on to the token. If this time expires then the token must be passed.

MAX_AC_4_ROTATION_TIME ::= VALUE_INTEGER_1

This variable can be written to and read but is not used in this implementation to perform a function and will have no effect on the MAC performance.

MAX_AC_2_ROTATION_TIME ::= VALUE_INTEGER_1

This variable can be written to and read but is not used in this implementation to perform a function and will have no effect on the MAC performance.

MAX_AC_0_ROTATION_TIME ::= VALUE_INTEGER_1

This variable can be written to and read but is not used in

this implementation to perform a function and will have no effect on the MAC performance.

MAC_RING_MAINTENANCE_ROTATION_TIME ::= VALUE_INTEGER_1

This variable can be written to and read but is not used in this implementation to perform a function and will have no effect on the MAC performance.

RING_MAINTENANCE_TIMER_INITIAL_VALUE ::= VALUE_INTEGER_1

This variable can be written to and read but is not used in this implementation to perform a function and will have no effect on the MAC performance.

MAX_INTER_SOLICIT_COUNT ::= VALUE_INTEGER_1

This variable can be written to and read but is not used in this implementation to perform a function and will have no effect on the MAC performance.

MIN_POST_SILENCE_PREAMBLE_LENGTH ::= VALUE_INTEGER_1

This variable can be written to and read but is not used in this implementation to perform a function and will have no effect on the MAC performance.

IN_RING_DESIRED ::= VALUE_INTEGER_1

This variable will allow the MAC to participate in the ring when set to one. When the command initially comes to set this variable to a one the token is kicked off (presumable for the first time). Sending multiple IN_RING_DESIRED commands of one will generate multiple tokens in the media. This is adeviation from IEEE 802.4, but the described usage of
this variable will be very
useful for testing. A negative one command will tell the MAC to consume the next tokens it receives (i.e. it will do nothing and not pass the token currently held). This will, of
course, lock up the media when the last token is consumed so at some point it will be necessary to set IN_RING_DESIRED to one again.

This work is implemented by setting force_token_pass to true and calling MAC_Transmit_queue() when the IN_RING_DESIRED = 1 or by sending a NULL work package to the interrupt routines when IN_RING_DESIRED = -1 (this will consume the current or next token). Keep in mind that this system will automatically attempt to consume extra tokens so the station manager may have to watch the dual token event reports if it really wishes to keep track of multiple tokens.

EVENT_ENABLE_MASK ::= EVENT_ENABLE_BITS

```
EVENT_ENABLE_BITS ::= BIT STRING
{ NS_CHANGED                          (0),
  NS_NULL                             (1),
  DUPLICATE_ADDRESS                   (2),
  FAULTY_TRANSMITTER                  (3),
  XMIT_QUEUE_THRESHOLD_EXCEEDED       (4),
  RECEIVE_QUEUE_THRESHOLD_EXCEEDED    (5),
  WATCH_DOG_TIMEOUT                   (6),
  FROZEN                              (7),
  TOKEN_LOST                          (8),
  DUAL_TOKEN                          (9),
  MAX_RETRY_ENCOUNTERED               (10) }
                        -- Where 1 is enabled
```

The MAC will report events when discovered and the appropriate bit is set in the MASK above. Bit 0 is the NS_Station, bit 1 is the NS_NULL etc. These bits are inspected each time the event has occurred and the MAC task is active. The event is reported only once whenever the actual occurence is detected.

MAX_RETRY_LIMIT ::= VALUE_INTEGER_1

This is the maximum number of times that a packet will be retransmitted when the acknowledgement indicates a bad transmittion. Since this is a connectionless system this variable should not be used. However, it will be loaded into the hardware transmit register whenever a packet is to be sent and the hardware will actually perform retrys.

MA_GROUP_ADDRESS ::= VALUE_ADDRESS_1

The MAC can respond to a group of group addresses. This is one of two methods for the Station Manager to tell the MAC which addresses are acceptable. This implementation will support a total of 16 group addresses. A positive value in

this command will set this address as part of the group addresses (unless there all used up) and a negative address will delete this address from the table.

MA_GROUP_ADDRESS_ALL ::= VALUE_ADDRESS_16

The MAC can respond to a group of group addresses. This is one of two methods for the Station Manager to tell the MAC which addresses are acceptable. This implementation will support a total of 16 group addresses. This command will write or read all 16 addresses at once. Addresses which are not valid addresses should be set to a negative one. Therefore only positive addresses will be passed in with this command unless there a negative one (a null address).

CHANNEL_ASSIGNMENTS ::= VALUE_INTEGER_1

This variable can be written to and read but is not used in this implementation to perform a function and will have no effect on the MAC performance.

TRANSMITTED_POWER_LEVEL_ADJUSTMENT ::= VALUE_INTEGER_1

This variable can be written to and read but is not used in this implementation to perform a function and will have no effect on the MAC performance.

TRANSMITTED_OUTPUT_INHIBITS ::= VALUE_INTEGER_1

This variable can be written to and read but is not used in this implementation to perform a function and will have no effect on the MAC performance.

RECEIVED_SIGNAL_SOURCES ::= VALUE_INTEGER_1

This variable can be written to and read but is not used in this implementation to perform a function and will have no effect on the MAC performance.

SIGNALING_MODE ::= VALUE_INTEGER_1

This variable can be written to and read but is not used in this implementation to perform a function and will have no effect on the MAC performance.

RECEIVED_SIGNAL_LEVEL_REPORTING ::= VALUE_INTEGER_1

This variable can be written to and read but is not used in this implementation to perform a function and will have no effect on the MAC performance.

LAN_TOPOLOGY_TYPE ::= VALUE_INTEGER_1

This variable can be written to and read but is not used in this implementation to perform a function and will have no effect on the MAC performance.

FREEZE_MAC ::= VALUE_INTEGER_1

This variable when set to one will freeze the MAC from taking any data from the local LLC. A negative one will unfreeze it and allow any queued messages to be processed. This will cause a burst effect and may cause loss of data due to the connectionless nature of the system and the limited buffer space.

MAC_TYPE ::= 04h

This variable is a read only variable and indicates which version of MAC is responding.

```
STATUS_TYPE ::= CHOICE
{     UNDEFINED_ERROR            [0]   VALUE_INTEGER_1 |
      SUCCESS                    [1]   VALUE_INTEGER_1 |
      REFUSE_TO_COMPLY           [2]   VALUE_INTEGER_1 |
      NOT_SUPPORTED              [3]   VALUE_INTEGER_1 |
      ERROR_IN_PERFOR            [4]   VALUE_INTEGER_1 |
      NOT_AVAILABLE              [5]   VALUE_INTEGER_1 |
      BAD_PARAMETER_ID           [6]   VALUE_INTEGER_1 |
      BAD_PARAMETER_OPERATION    [7]   VALUE_INTEGER_1 |
      BAD_PARAMETER_VALUE        [8]   VALUE_INTEGER_1 |
      BAD_EXPECTED_VALUE         [9]   VALUE_INTEGER_1  }
```

These are responses to a command indicating the status of the command. Following are expected uses of these responses;

```
UNDEFINED_ERROR - Request was not understood or no
                  appropriate error message available.
SUCCESS - A successful operation has been completed.
REFUSE_TO_COMPLY - The operation was impossible or illegal.
NOT_SUPPORTED - The operation is not supported or
                recognized.
ERROR_IN_PERFOR - A error was encountered
                  during operation.
NOT_AVAILABLE - Information is not yet
                available.
BAD_PARAMETER_ID - Parameter ID was not
                   recognized.
BAD_PARAMETER_OPERATION - Operation requested
                          was not recognized
BAD_PARAMETER_VALUE - The Parameter
                      value was bad.
BAD_EXPECTED_VALUE  - The expected value was
                      illegal.
```

```
EVENT_TYPES ::= IMPLICIT SEQUENCE
{ EVENT_CLASS          EVENT_CLASS_TYPES      }

     EVENT_CLASS_TYPES ::=  CHOICE
     { LOCAL       [0]    EVENT_IDENTIFIER_TYPES      |
       REMOTE      [1]    EVENT_IDENTIFIER_TYPES       }
```

Events in this implementation are always LOCAL (as opposed to events that occurred in a remote node).

```
EVENT_IDENTIFIER_TYPES ::= CHOICE
{ NS_CHANGED                          [0]  VALUE_INTEGER_1   |
```

```
  NS_NULL                              [1]  VALUE_INTEGER_1   |
  DUPLICATE_ADDRESS                    [2]  VALUE_INTEGER_1   |
  FAULTY_TRANSMITTER                   [3]  VALUE_INTEGER_1   |
  XMIT_QUEUE_THRESHOLD_EXCEEDED        [4]  VALUE_INTEGER_1   |
  RECEIVE_QUEUE_THRESHOLD_EXCEEDED     [5]  VALUE_INTEGER_1   |
  WATCH_DOG_TIMEOUT                    [6]  VALUE_INTEGER_1   |
  FROZEN                               [7]  VALUE_INTEGER_1   |
  TOKEN_LOST                           [8]  VALUE_INTEGER_1   |
  DUAL_TOKEN                           [9]  VALUE_INTEGER_1   |
  MAX_RETRY_ENCOUNTERED                [10] VALUE_INTEGER_1   |
  BAD_MESSAGE_SENT                     [11] VALUE_ADDRESS_1   }
```

These events are reported upon the discovery of the following conditions;

NS_CHANGED - Flagged when the event routine discovers a change in the NS address.

NS_NULL - Flagged when the NS is set to NULL

DUPLICATE_ADDRESS - Does nothing since FODs doesn't report other addresses.

FAULTY_TRANSMITTER - Does nothing since FODs doesn't report bad transmitters.

XMIT_QUEUE_THRESHOLD_EXCEEDED - Flagged when the MAC cannot get buffer space for outgoing data.

RECEIVE_QUEUE_THRESHOLD_EXCEEDED - Flagged when the MAC cannot get buffer space for incoming data.

WATCH_DOG_TIMEOUT - Flagged if the hardware watch dog timer expires.

FROZEN - Flagged when the MAC is frozen. Reported only once.

TOKEN_LOST - Flagged when the token is detected as lost.

DUAL_TOKEN - Flagged when a extra token is discovered.

MAX_RETRY_ENCOUNTERED - Flagged when a the max retry is encountered. Any IRL4 interrupt indicates the hardware retryed

beyond the retry limit.

BAD_MESSAGE_SENT -Flagged when the MAC discovers a message which does not agrees with its indicated structure size (i.e. bad length field).

```
ACTION_VALUE_TYPES ::= CHOICE
{ RESET              [0]   VALUE_INTEGER_1          |
  FREEZE/UNFREEZE    [1]   FREEZE_MAC               |
  ENTER_THE_RING     [2]   IN_RING_DESIRED            }
```

The ACTION_VALUE_TYPES allow the following;

Reset value_integer_1 = anything:

A reset will flush all queues, set all operating parameters to their initial values, lose the token (if its holding it), and await work from either the media or the LLC.

FREEZE/UNFREEZE FREEZE_MAC = 1 will freeze mac.
= -1 will unfreeze mac.

A FREEZE/UNFREEZE command with Freeze option will make the MAC main task ignore all queues but the SM. In effect the MAC is frozen to local service only. Packets arriving from remote nodes and from the LLC will be queued until the buffer is exceeded or the MAC is unfrozen. The token is still passed as normal. Commands from the Station Manager are processed while frozen.

ENTER_THE_RING IN_RING_DESIRED = 1 create token
= -1 consume token

A ENTER_THE_RING command will convince the MAC it is the holder of the token. If this command is sent twice in a row then the MAC will;

1) In the case of the MAC currently holding the token, do nothing.

2) In the case of the MAC about ready to receive the token, a dual token may be reported and the extra token is consumed (an inherent operation in this design).

3) In the case of the MAC in a idle state, the token is taken and passed on when appropriate. Since this is a CSMA/CD machine it can operate with two tokens for

some time before it discovers and consumes one of them (802.4 machines also have this ability but are far less forgiving of collisions).

```
OPERATION_COMMAND_TYPES ::= CHOICE
  {TEST_<<              [0]  READ_WRITE_VALUE_TYPES    |
   TEST_>>              [1]  READ_WRITE_VALUE_TYPES    |
   TEST_==              [2]  READ_WRITE_VALUE_TYPES    |
   TEST_<>              [3]  READ_WRITE_VALUE_TYPES    |
   TEST_<=              [4]  READ_WRITE_VALUE_TYPES    |
   TEST_>=              [5]  READ_WRITE_VALUE_TYPES    |
   <<_GIVEN_CONSTANT    [6]  GIVEN                     |
   >>_GIVEN_CONSTANT    [7]  GIVEN                     |
   ==_GIVEN_CONSTANT    [8]  GIVEN                     |
   <>_GIVEN_CONSTANT    [9]  GIVEN                     |
   <=_GIVEN_CONSTANT    [10] GIVEN                     |
   >=_GIVEN_CONSTANT    [11] GIVEN                       }
```

The above operations expects a variable (we'll call var1) to be internal. The complete structure includes either a variable or constant which we'll call var2. The constant is used to overwrite Var1 in case the operation test true so in the case of two internal vars being tested a constant is also passed in. The above operation commands imply the following:

```
TEST_<<  - if var1 << var2 then var1=constant
TEST_>>  - if var1 >> var2 then var1=constant
TEST_==  - if var1 == var2 then var1=constant
TEST_<>  - if var1 <> var2 then var1=constant
TEST_<=  - if var1 <= var2 then var1=constant
TEST_>=  - if var1 >= var2 then var1=constant
<<_GIVEN_CONSTANT - if var1 << constant then var1=constant
>>_GIVEN_CONSTANT - if var1 >> constant then var1=constant
==_GIVEN_CONSTANT - if var1 == constant then var1=constant
<>_GIVEN_CONSTANT - if var1 <> constant then var1=constant
<=_GIVEN_CONSTANT - if var1 <= constant then var1=constant
>=_GIVEN_CONSTANT - if var1 <= constant then var1=constant
```

Var1 is a MAC parameter to be tested (internal). Its value is always returned along with a status. Var2 is a MAC parameter (internal) or a constant (external) used in the comparison of Var1 (internal). Var1 always refers to a variable located in the MAC. Var2 is either located in the MAC (a compare of two internal variables) or as a constant (external) passed in. In all cases a true test forces Var1 to be a external constant.

```
CONSTANT ::=  VALUE_INTEGER_1
```

```
VALUE_INTEGER_1 ::= IMPLICIT LONG_WORD

VALUE_ADDRESS_1 ::=  IMPLICIT LONG_WORD (32 BITS)

VALUE_ADDRESS_16 ::=  IMPLICIT ARRAY OF 16 LONG_WORDS
                                (32 BITS EACH)
```

4.2.6 SYNTAX -
STATION MANAGER INTERFACE SYNTAX

The station manager communicates to the MAC across the MII. The syntax of such communication is described below according to Abstract Syntax Notation One or ASN.1 (ISO DIS 8824). The information described is encoded to the basic coding rules as found in ASN.1 (ISO DIS 8825). Some sample records follow the syntax notations.

4.2.7 FORMAL SYNTAX SPECIFICATION -

```
MESSAGE_RECORD ::= [PRIVATE 0] CHOICE
{ [0]  SM_MAC_LM_SET_VALUE.INVOKE                          |
  [1]  SM_MAC_LM_SET_VALUE.REPLY                           |
  [2]  SM_MAC_LM_GET_VALUE.INVOKE                          |
  [3]  SM_MAC_LM_GET_VALUE.REPLY                           |
  [4]  SM_MAC_LM_COMPARE_AND_SET_VALUE.INVOKE              |
  [5]  SM_MAC_LM_COMPARE_AND_SET_VALUE.REPLY               |
  [6]  SM_MAC_ACTION_VALUE.INVOKE                          |
  [7]  SM_MAC_ACTION_VALUE.REPLY                           |
  [8]  SM_MAC_EVENT_VALUE.NOTIFY                           |
  [9]  SM_MAC_EVENT_VALUE.REPLY                           }


SM_MAC_LM_SET_VALUE.INVOKE ::= IMPLICIT SEQUENCE
{ PARAMETER_TYPE          READ_WRITE_VALUE_TYPES ,
  ACCESS_CONTROL_INFO     NULL                   }

SM_MAC_LM_SET_VALUE.REPLY ::= IMPLICIT SEQUENCE
{ RETURN_VAL          READ_WRITE_VALUE_TYPES,
  STATUS                   STATUS_TYPE}

SM_MAC_LM_GET_VALUE.INVOKE ::= IMPLICIT SEQUENCE
{ PARAMETER_TYPE          READ_WRITE_VALUE_TYPES ,
  ACCESS_CONTROL_INFO     NULL                   }

SM_MAC_LM_GET_VALUE.REPLY ::= IMPLICIT SEQUENCE
{ PARAMETER_TYPE     READ_WRITE_VALUE_TYPES ,
  STATUS             STATUS_TYPE }


SM_MAC_LM_COMPARE_AND_SET_VALUE.INVOKE ::= IMPLICIT SEQUENCE
{ PARAMETER_TYPE          DUMMY_RW_TYPES,
  OPERATION_COMMAND       OPERATION_COMMAND_TYPES,
  ACCESS_CONTROL_INFO     NULL                   }

SM_MAC_LM_COMPARE_AND_SET_VALUE.REPLY ::= IMPLICIT SEQUENCE
{  RETURN_VAL             READ_WRITE_VALUE_TYPES,
   STATUS                 STATUS_TYPE }


SM_MAC_ACTION_VALUE.INVOKE ::= IMPLICIT SEQUENCE
{ PARAMETER_ID               ACTION_VALUE_TYPES ,
  ACCESS_CONTROL_INFO        NULL}

SM_MAC_ACTION_VALUE.REPLY ::= IMPLICIT SEQUENCE
{ STATUS                  STATUS_TYPE,
  ACTION_REPORT           NULL     }
```

```
ACTION_VALUE_TYPES ::= CHOICE
{ RESET                [0]   VALUE_INTEGER_1          |
  FREEZE/UNFREEZE      [1]   FREEZE_MAC               |
  ENTER_THE_RING       [2]   IN_RING_DESIRED            }


OPERATION_COMMAND_TYPES ::= CHOICE
  {TEST_<<             [0]   DUMMY_RW_TYPES           |
   TEST_>>             [1]   DUMMY_RW_TYPES           |
   TEST_==             [2]   DUMMY_RW_TYPES           |
   TEST_<>             [3]   DUMMY_RW_TYPES           |
   TEST_<=             [4]   DUMMY_RW_TYPES           |
   TEST_>=             [5]   DUMMY_RW_TYPES           |
   <<_GIVEN_CONSTANT   [6]   GIVEN                    |
   >>_GIVEN_CONSTANT   [7]   GIVEN                    |
   ==_GIVEN_CONSTANT   [8]   GIVEN                    |
   <>_GIVEN_CONSTANT   [9]   GIVEN                    |
   <=_GIVEN_CONSTANT   [10]  GIVEN                    |
   >=_GIVEN_CONSTANT   [11]  GIVEN                     }


GIVEN ::= CHOICE
  { [0] VALUE_INTEGER_1      |
    [1] VALUE_ADDRESS_1       }

CONSTANT ::=  VALUE_INTEGER_1

VALUE_INTEGER_1 ::= IMPLICIT INTEGER

VALUE_ADDRESS_1 ::=  IMPLICIT LONG_WORD (32 BITS)

VALUE_ADDRESS_16 ::=  IMPLICIT ARRAY OF 16 LONG_WORDS
                                (32 BITS EACH)
```

```
MAC_SM_EVENT_VALUE.NOTIFY ::= IMPLICIT SEQUENCE
{ EVENT_ID                    EVENT_TYPES }

MAC_SM_EVENT_VALUE.REPLY ::= IMPLICIT SEQUENCE
{ STATUS                      STATUS_TYPE }
```

```
READ_WRITE_VALUE_TYPES ::=  CHOICE        {
   [0]       MAC_TYPE                                  |
   [1]       NS                                        |
   [2]       SLOT_TIME                                 |
   [3]       HI_PRI_TOKEN_HOLD_TIME                    |
   [4]       MAX_AC_4_ROTATION_TIME                    |
   [5]       MAX_AC_2_ROTATION_TIME                    |
   [6]       MAX_AC_0_ROTATION_TIME                    |
   [7]       MAC_RING_MAINTENANCE_ROTATION_TIME        |
   [8]       RING_MAINTENANCE_TIMER_INITIAL_VALUE      |
   [9]       MAX_INTER_SOLICIT_COUNT                   |
   [10]      MIN_POST_SILENCE_PREAMBLE_LENGTH          |
   [11]      EVENT_ENABLE_MASK                         |
   [12]      MAX_RETRY_LIMIT                           |
   [13]      MA_GROUP_ADDRESS                          |
   [14]      MA_GROUP_ADDRESS_ALL                      |
   [15]      CHANNEL_ASSIGNMENTS                       |
   [16]      TRANSMITTED_POWER_LEVEL_ADJUSTMENT        |
   [17]      TRANSMITTED_OUTPUT_INHIBITS               |
   [18]      RECEIVED_SIGNAL_SOURCES                   |
   [19]      SIGNALING_MODE                            |
   [20]      RECEIVED_SIGNAL_LEVEL_REPORTING           |
   [21]      LAN_TOPOLOGY_TYPE                         |
   [22]      TS                                        }
```

```
DUMMY_RW_TYPES ::=  CHOICE        {
MAC_TYPE                                    [0] VALUE_INTEGER_1   |
NS                                          [1] VALUE_INTEGER_1   |
SLOT_TIME                                   [2] VALUE_INTEGER_1   |
HI_PRI_TOKEN_HOLD_TIME                      [3] VALUE_INTEGER_1   |
MAX_AC_4_ROTATION_TIME                      [4] VALUE_INTEGER_1   |
MAX_AC_2_ROTATION_TIME                      [5] VALUE_INTEGER_1   |
MAX_AC_0_ROTATION_TIME                      [6] VALUE_INTEGER_1   |
MAC_RING_MAINTENANCE_ROTATION_TIME          [7] VALUE_INTEGER_1   |
RING_MAINTENANCE_TIMER_INITIAL_VALUE        [8] VALUE_INTEGER_1   |
MAX_INTER_SOLICIT_COUNT                     [9] VALUE_INTEGER_1   |
MIN_POST_SILENCE_PREAMBLE_LENGTH           [10] VALUE_INTEGER_1   |
EVENT_ENABLE_MASK                          [11] VALUE_INTEGER_1   |
MAX_RETRY_LIMIT                            [12] VALUE_INTEGER_1   |
MA_GROUP_ADDRESS                           [13] VALUE_INTEGER_1   |
MA_GROUP_ADDRESS_ALL                       [14] VALUE_INTEGER_1   |
CHANNEL_ASSIGNMENTS                        [15] VALUE_INTEGER_1   |
TRANSMITTED_POWER_LEVEL_ADJUSTMENT         [16] VALUE_INTEGER_1   |
TRANSMITTED_OUTPUT_INHIBITS                [17] VALUE_INTEGER_1   |
RECEIVED_SIGNAL_SOURCES                    [18] VALUE_INTEGER_1   |
SIGNALING_MODE                             [19] VALUE_INTEGER_1   |
RECEIVED_SIGNAL_LEVEL_REPORTING            [20] VALUE_INTEGER_1   |
LAN_TOPOLOGY_TYPE                          [21] VALUE_INTEGER_1   |
TS                                         [22] VALUE_INTEGER_1   }
```

TS ::= VALUE_ADDRESS_1

NS ::= VALUE_ADDRESS_1

SLOT_TIME ::= VALUE_INTEGER_1

HI_PRI_TOKEN_HOLD_TIME ::= VALUE_INTEGER_1

MAX_AC_4_ROTATION_TIME ::= VALUE_INTEGER_1

MAX_AC_2_ROTATION_TIME ::= VALUE_INTEGER_1

MAX_AC_0_ROTATION_TIME ::= VALUE_INTEGER_1

MAC_RING_MAINTENANCE_ROTATION_TIME ::= VALUE_INTEGER_1

RING_MAINTENANCE_TIMER_INITIAL_VALUE ::= VALUE_INTEGER_1

MAX_INTER_SOLICIT_COUNT ::= VALUE_INTEGER_1

MIN_POST_SILENCE_PREAMBLE_LENGTH ::= VALUE_INTEGER_1

IN_RING_DESIRED ::= VALUE_INTEGER_1

EVENT_ENABLE_MASK ::= EVENT_ENABLE_BITS

MAX_RETRY_LIMIT ::= VALUE_INTEGER_1

MA_GROUP_ADDRESS ::= VALUE_ADDRESS_1

MA_GROUP_ADDRESS_ALL ::= VALUE_ADDRESS_16

CHANNEL_ASSIGNMENTS ::= VALUE_INTEGER_1

TRANSMITTED_POWER_LEVEL_ADJUSTMENT ::= VALUE_INTEGER_1

TRANSMITTED_OUTPUT_INHIBITS ::= VALUE_INTEGER_1

RECEIVED_SIGNAL_SOURCES ::= VALUE_INTEGER_1

SIGNALING_MODE ::= VALUE_INTEGER_1

RECEIVED_SIGNAL_LEVEL_REPORTING ::= VALUE_INTEGER_1

LAN_TOPOLOGY_TYPE ::= VALUE_INTEGER_1

FREEZE_MAC ::= VALUE_INTEGER_1

MAC_TYPE ::= 04h

```
STATUS_TYPE ::= CHOICE
{     UNDEFINED_ERROR              [0]   VALUE_INTEGER_1 |
      SUCCESS                      [1]   VALUE_INTEGER_1 |
      REFUSE_TO_COMPLY             [2]   VALUE_INTEGER_1 |
      NOT_SUPPORTED                [3]   VALUE_INTEGER_1 |
      ERROR_IN_PREFOR              [4]   VALUE_INTEGER_1 |
      NOT_AVAILABLE                [5]   VALUE_INTEGER_1 |
      BAD_PARAMETER_ID             [6]   VALUE_INTEGER_1 |
      BAD_PARAMETER_OPERERATION    [7]   VALUE_INTEGER_1 |
      BAD_PARAMETER_VALUE          [8]   VALUE_INTEGER_1 |
      BAD_EXPECTED_VALUE           [9]   VALUE_INTEGER_1  }

EVENT_TYPES ::= IMPLICIT SEQUENCE
{ EVENT_CLASS          EVENT_CLASS_TYPES      }

EVENT_CLASS_TYPES ::=  CHOICE
{ LOCAL       [0]     EVENT_IDENTIFIER_TYPES          |
  REMOTE      [1]     EVENT_IDENTIFIER_TYPES              }

EVENT_IDENTIFIER_TYPES ::= CHOICE
{ NS_CHANGED                           [0]  VALUE_INTEGER_1   |
  NS_NULL                              [1]  VALUE_INTEGER_1   |
  DUPLICATE_ADDRESS                    [2]  VALUE_INTEGER_1   |
  FAULTY_TRANSMITTER                   [3]  VALUE_INTEGER_1   |
  XMIT_QUEUE_THRESHOLD_EXCEEDED        [4]  VALUE_INTEGER_1   |
  RECEIVE_QUEUE_THRESHOLD_EXCEEDED     [5]  VALUE_INTEGER_1   |
  WATCH_DOG_TIMEOUT                    [6]  VALUE_INTEGER_1   |
  FROZEN                               [7]  VALUE_INTEGER_1   |
  TOKEN_LOST                           [8]  VALUE_INTEGER_1   |
  DUAL_TOKEN                           [9]  VALUE_INTEGER_1   |
  MAX_RETRY_ENCOUNTERED                [10] VALUE_INTEGER_1   |
  BAD_MESSAGE_SENT                     [11] VALUE_ADDRESS_1    }


EVENT_ENABLE_BITS ::= BIT STRING
{ NS_CHANGED                           (0),
  NS_NULL                              (1),
  DUPLICATE_ADDRESS                    (2),
  FAULTY_TRANSMITTER                   (3),
  XMIT_QUEUE_THRESHOLD_EXCEEDED        (4),
  RECEIVE_QUEUE_THRESHOLD_EXCEEDED     (5),
  WATCH_DOG_TIMEOUT                    (6),
  FROZEN                               (7),
  TOKEN_LOST                           (8),
  DUAL_TOKEN                           (9),
  MAX_RETRY_ENCOUNTERED                (10)
  BAD_MESSAGE_SENT                     (11) }
                        -- Where 1 is enabled
```